Optimization of energy and power driven architectural exploration for multi-core and heterogeneous System on Chip

Von der Carl-Friedrich-Gauß-Fakultät

der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von

Syed Abbas Ali SHAH

geboren am 20.02.1982

in Mansehra, Pakistan

Eingereicht am: 10.04.2018

Disputation am: 19.12.2018

Referent: Prof. Dr. Ing. Mladen Berekovic

Koreferent: Prof. Dr. Ing. Harald Michalik

(2019)

Abstract

Continuously growing computational complexity observed in all embedded systems domains require highly optimized and well targeted SoC designs. These designs are typically composed of many processing cores coupled with the specialized hardware accelerators to achieve higher computational throughput at lower power dissipation. Traditionally design tasks were executed sequentially, software driver and application development start only after the actual hardware becomes available. However, the first availability of silicon samples takes a substantial amount of time and increasingly delays the software development. This strong dependency in the production cycle has very serious consequences on time-to-market demands. To optimize hardware software codesign, academia and industry propose virtual prototyping as a multifaceted solution. A virtual prototype emulates the behavior of the target hardware architecture, which aids early architecture exploration as well as software development in parallel to hardware finalization and production. Recently, virtual platforms started to utilize the SystemC/TLM standards to provide reusable system-level hardware component models on different abstraction levels; a cycle-accurate model for higher accuracy and functional-accurate for higher simulation performance. These models allow engineers to flexibly evaluate different architectural parameter configurations to meet the target application's requirements from early on. Within this exploration, the design process of efficient SoCs involves multiple iterations; starting with the initial set of parameters, to be repeatedly adjusted towards the desired design goal. The contribution of this work builds on top of the established virtual prototype platforms to improve both SoC design quality and productivity. Initially, an automatic system-level power estimation framework was developed to address the critical issue of early power estimation in SoC design. The estimation framework models the static and dynamic power consumption of the hardware components. These system-level estimation models are created from the normalized values of the basic design components of SoC, obtained through one-time power simulation of accurate hardware models. The framework allows dynamic technology node reconfiguration for power estimation models and integration of new power models for third party SoC components. Its instantaneous power reporting through a graphical user interface further aids the detection of possible hotspot early into the design process. Adding this additional data in conjunction with a steadily growing design space of complex heterogeneous SoC, finding the right parameter configuration is a challenging and laborious task for a system-level designer. This work addresses this bottleneck by optimizing the design space exploration (DSE) process for MPSoC design. An automatic DSE framework for virtual platforms was developed which is flexible and allows the selection optimal parameter configuration without pre-existing knowledge. To reduce exploration time, the framework is equipped with several multi-objective optimization techniques based on simulated annealing and a genetic algorithm inspired by evolutionary computing. Lastly, to aid hardware/software partitioning at system-level, a flexible and automated workflow is presented. The workflow allows the designer to explore various possible partitioning scenarios without going into depth of the hardware architecture complexity and software integration. The

framework generates system-level hardware accelerators from corresponding functionality encoded in the software code and integrates them into the virtual platform. Detail data in terms of power consumption, speedup, and communication overhead of acceleration is captured and reported to the designer, which further increases the quality and productivity of the development process towards the final architecture. The presented tools are evaluated using a state-of-the-art virtual platform for a range of single and multi-core applications. Viewing the energy delay product (EDP), a reduction in exploration time was recorded at approximately 62% (worst case), maintaining optimal parameter accuracy of 90% compared to previous techniques. While the automated hardware/software partitioning workflow further increases the exploration versatility by combining modern high-level synthesis with system-level architectural exploration.

Zusammenfassung

Die ständig wachsende Komplexität in allen Domänen der eingebetteten Systeme erfordert hoch optimierte und zielgerichtete SoC-Designs. Diese bestehen typischerweise aus vielen Prozessorkernen, die um spezialisierte Hardwarebeschleunigern ergänzt werden, um den Rechendurchsatz bei geringerer Verlustleistung steigern zu können. Herkömmlicherweise erfolgt der angewendete Entwicklungsprozess sequentiell, die Entwicklung von Treibersoftware und Applikationen erfolgt erst nachdem die tatsächliche Hardware verfügbar ist. Die benötigten Hardwareprototypen werden allerdings erst nach einer beträchtlichen Zeit verfügbar, wodurch die Softwareentwicklung zusätzlich verzögert wird. Diese starke Abhängigkeit vom Entwicklungsschritten hat erhebliche Konsequenzen für die termingerechte Markteinführung. Um die parallele Entwicklung von Hardware und Software zu optimieren, wurde das Konzept der virtuellen Prototypen aus Wissenschaft und Industrie als vielseitiges Werkzeug hervorgebracht. Ein virtueller Prototyp emuliert das Verhalten der Zielhardwarearchitektur, die sowohl die frühe Architekturevaluation als auch die Softwareentwicklung parallel zur Architektur Finalisierung und Produktion unterstützt. Moderne virtuelle Prototypenplattformen beginnen vermehrt die SystemC/TLM-Standards zu verwenden, was die Wiederverwendbarkeit von Hardwarekomponentenmodellen auf der Systemebene erlaubt. Diese Modelle werden typischerweise auf verschiedenen Abstraktionsebenen bereitzustellen; ein zyklusgenaues Modell mit einer hohen Genauigkeit und ein funktionsgenaues Modell mit gesteigerter Simulationsleistung. Diese Modelle ermöglichen es den Ingenieuren verschiedene Architekturparameterkonfigurationen flexibel zu testen, um die funktionale und nichtfunktionale Anforderungen der Zielanwendung frühzeitig untersuchen zu können. In diesem iterativen Prozess werden Parameterkonfigurationen wiederholt angepasst und per Simulation bewertet um die am besten geeignete Architektur zu identifizieren. Der Beitrag dieser Arbeit baut auf dem etablierten Konzept der virtuellen Prototyp Plattformen auf, um die Qualität und die Produktivität des Entwurfsprozesses zu verbessern. Zunächst wurde ein automatisches System-Level-Framework entwickelt, um Verlustleistungsabschätzung für SoC-Designs in einer deutlich früheren Entwicklungsphase zu ermöglichen. Hierfür werden statischen und dynamischen Energieverbrauchsanteile individueller Hardwarekomponenten durch ein abstraktes Modell ausgedrückt. Die zugrundeliegenden Energieverbrauchseigenschaften werden hierfür aus hochgenauen Hardwaremodellen extrahiert und normiert in ein allgemeines Modell überführt. Das Framework ermöglicht eine dynamische Anpassung des Technologieknotens in den Leistungsschätzungsmodellen sowie die Integration neuer Leistungsmodelle für SoC-Komponenten von Drittanbietern. Die kontinuierliche Erfassung der Energieverbrauchseigenschaften und ihre Darstellung auf einer grafischen Benutzeroberfläche unterstützt zusätzlich die frühzeitige Identifikation möglicher Hotspots. Durch die Bereitstellung zusätzlicher Daten, in Verbindung mit einem stetig wachsenden Entwurfsraum komplexer heterogener SoCs, ist die Identifikation der richtigen Parameterkonfiguration eine komplexe und zeitintensive Aufgabe. Die vorgelegte Arbeit behandelt diese Problemstellung durch eine gesteigerte Automatisierung des DSE-Prozesses

(Design Space Exploration). Es wurde ein automatischer Ablauf für virtuelle Plattformen entwickelt, der flexibel die Auswahl einer optimalen Parameterkonfiguration ohne vorheriges Wissen oder Erfahrungen ermöglicht. Hierbei werden Techniken der mehrdimensionalen Optimierung, basierend auf Simulated Annealing und genetischer Algorithmen, angewendet. Schließlich wurde zur Unterstützung der Hardware/Software-Partitionierung auf System-Ebene ein flexibler und automatisierter Workflow entwickelt. Er ermöglicht es dem Designer verschiedene mögliche Partitionierungsszenarien zu untersuchen, ohne sich in die Komplexität der Hardwarearchitektur und der Softwareintegration zu vertiefen. Das Framework erzeugt System-Level-Hardware-Beschleuniger aus korrespondierenden Funktionen des Softwarequellcodes und integriert sie in nahtlos die ausführbare virtuelle Plattform. Detaillierte Daten zum Energieverbrauch, Beschleunigungsfaktor und Kommunikationsoverhead der Partitionierung werden erfasst und dem Designer zur Verfügung gestellt, was die Qualität und Produktivität des weiter erhöht. Die vorgestellten Tools werden mit einer modernen virtuellen Plattform für eine Reihe von Single- und Multi-Prozessor-Anwendungen evaluiert. Bei Betrachtung des Energieverzögerungsprodukts (EDP) wurde eine Verringerung der Explorationszeit um ca. 62% (schlimmster Fall) festgestellt, während eine Parametergenauigkeit von 90% im Vergleich zu früheren Techniken erhalten bleibt. Darauf aufbauend, erleichtert die automatisierte Untersuchung verschiedener Hardware/Software Partitionierungen die Entwicklung heterogener Architekturen durch die Kombination moderner High-Level-Synthese mit Architektur-Exploration auf der Systemebene.

Acknowledgements

I would like to thank my advisor Prof. Mladen Berekovic for his continuous support, guidance and encouragement for my Ph.D. research. His immense knowledge and prompt feedback helped me grow as a research scientist.

I am thankful to the SoCRocket team and my colleagues: Thomas Schuster, Rolf Meyer, Jan Wagner, Bastian Farkas, Ioanna Tsekoura, Parvin Bahmanyar, Shaodong Qin, Sönke Michalik, Sören Michalik, Azaam Mahmood, Jamin Naghmouchi and Amel Alfahham who played a vital role in my doctoral path.

During my stay in Braunschweig, I have made many friends. I would like to mention Zahid, Zeshan and Hassan who made this small city my second home. Hayder my friend and officemate, I would like to thank you, we shared the same office throughout our Ph.D. We had countless very interesting discussions which always refreshed my mind. My special thanks to my friend and colleague Sven with whom I shared most of my research ideas and his honest feedback steered me in the right direction. I would like to thank my friend Sikandar for his support and sharing ideas from computer vision which helped me to understand and align myself with the cross-field research. I would like to thank my wife and best friend Gosia for her support and sharing with the ups and downs during my Ph.D. research.

I would like to thank my siblings: Shahida, Shaukat, Rashida, Liaqat, Robab, and Shahzad for their love, unwavering support and encouragement. I am greatly indebted to my Mom for always being there whenever I needed. Your love and prayer helped me throughout my life. Finally, I cannot describe my gratitude to my father (R.I.P) in words, he would have been extremely proud to see me crossing the finishing line, the journey which I had started by holding his hand.

List of Publications

- S. A. A. Shah, S. Horsinka, B. Farkas, R. Meyer, and M. Berekovic: *Automatic Exploration of Hardware/Software Partitioning*, Design and Verification Conference (DVCon) United States 2017, February 27 - March 2, 2017 , San Jose, CA USA, 2017

- R. Meyer, B. Farkas, S. A. A. Shah, and M. Berekovic:: *Transparent SystemC Model Factory for Scripting Languages*, Design and Verification Conference (DVCon) United States 2017, February 27 - March 2, 2017 , San Jose, CA USA, 2017

- S. A. A. Shah, B. Farkas, R. Meyer, and M. Berekovic: *Accelerating MPSoC Design Space Exploration Within System-Level Frameworks*, The IEEE Nordic Circuits and Systems Conference (NORCAS), 1-2 November 2016 Copenhagen, Denmark, 2016

- B. Farkas, S. A. A. Shah, J. Wagner, R. Meyer, R. Buchty, and M. Berekovic: *An Open and Flexible SystemC to VHDL Workflow for Rapid Prototyping*, Design and Verification Conference (DVCon) Europe 2016, October 19 - 20, 2016 Munich, Germany, 2016

- H. Al-Khalissi, S. A. A. Shah, and M. Berekovic: *An Efficient Barrier Implementation for OpenMP-Like Parallelism on the Intel SCC*, PDP '14: Proceedings of the 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE Computer Society, 2014, ISBN 978-1-4799-2729-6

- S. A. A. Shah, J. Wagner, T. Schuster, and M. Berekovic: *A lightweight-system-level power and area estimation methodology for application specific instruction set processors Power and Timing Modeling*, Optimization and Simulation (PATMOS), 2014 24th International Workshop on, 2014

Contents

Abbreviations

AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Application programming interface
ASICs	Application Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
AT	Approximately Timed
BMP	Bitmap
CCSDS	Consultative Committee for Space Data Systems
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CSV	Comma- Separated Values
DCT	Discrete Cosine Transformation
DMA	Direct Memory Access
DMI	Direct Memory Interface
DSE	Design Space Exploration
DSP	Digital signal processor
EDA	Electronic design automation
EDP	Engegy Delay Product
EP	Exploration Prototype
ES	Exhaustive Search
ESA	European Space Agency
ESD	Electronic System Design
ESL	Electronic System Level
ESLD	Electronic System Level Design
FGS	Fast Genetic Search

FMPO	Fast Multi Parameter Optimization
FPGA	Field-programmable gate array
GDB	GNU Debugger
GPU	Graphics processing unit
GUI	Graphical user interface
HDL	Hardware description language
HLS	High-level synthesis
HW	Hardware
IF	Interface
IO	Input/output
IP	Intellectual Property
IRQ	Interrupt Request
ISS	Instruction Set Simulator
JSON	JavaScript Object Notation
LII	Language Interface Independent
LT	Loosely Timed
MMU	Memory management unit
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MPO	Multi Parameter Optimization
MPSoC	Multiprocessor System on Chip
NMOS	N-channel Metal Oxide Semiconductor
NoC	Network on Chip
OCP	Open Core Protocol
PCIe	Peripheral Component Interconnect Express
PNG	Portable Network Graphics
PROM	Programmable Read-Only Memory
PV	Programming View
RND	Random Optimization
ROM	Read Only memory
RTEMS	Real-Time Executive for Multiprocessor Systems
RTL	Register Transfer Level
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System on Chip
SPO	Single Parameter Optimization

SRAM	Static Random Access Memory
SW	Software
SWIG	Simplified Wrapper and Interface Generator
TCL	Tool Command Language
TLM	Transaction-level modeling
UART	Universal Asynchronous Receiver Transmitter
UML	Unified Modeling Language
USI	Universal Scripting Interface
VCD	Value change dump
VHDL	Very high speed Hardware Description language
VP	Virtual Prototype
XML	Extensible Markup Language

Dedicated to my sister

Syeda Phool Rukhsana

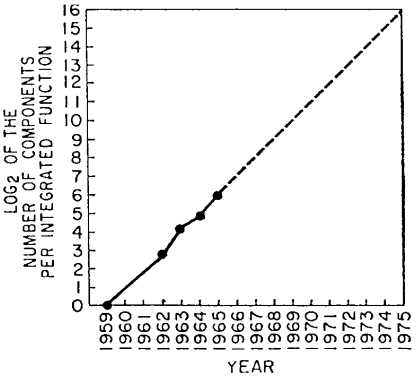
(Rest In Peace)

Chapter 1

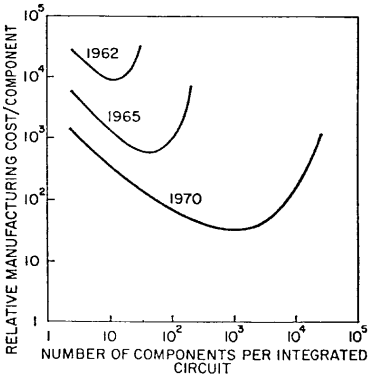
Introduction

The ubiquity of complex processing architectures in all computational domains drives progress at an unrepresented pace. While not exclusive anymore, embedded systems were the first and central point of origin for highly integrated systems. These are customarily composed of computational units (e.g. microcontrollers/microprocessors) and previously discrete custom peripheral components, integrated into a single chip design. Enabling the execution of extensive firmware, software applications ranging up to complex operating systems, which in hand need to be developed in conjunction with the hardware. Observing strict functional and non-functional requirements, such as low cost (should be affordable), short time-to-market (must available on time), real-time behavior for safety applications, high-performance, and low-power to increase the battery life or manage heat dissipation.

The traditional application domains of embedded systems are aerospace, agriculture, medical, entertainment industry, defense and many others. With the emergence of new technological fields such as machine learning, big data processing, mobility, security, robots, and Internet of Things (IoT), the embedded system market is growing at a rapid rate. According to the global research study ”2017 Retail Vision Study” [1] 70% of the retailers are planning to invest in the IoTs by the year 2021. The reason for this accelerated expansion of electronic devices is Moore/Dennard scaling, which was driving force behind the electronic manufacturing industry to meet the requirements of modern devices. However, since 2003 [2], the design complexity of modern embedded systems has



(a) Doubling components per integrated circuit prediction



(b) Model for Cost/Components and number of components/integrated circuit

Figure 1.1: Gordon Moore's predictions from 1965 [3]

risen significantly, which lead the industry to counter new design challenges in the absence of Moore expansion (e.g., need to design better hardware/software systems). In the following sections, the reasons for Moore/Dennard scaling failure, new design challenges and contributions of this thesis are discussed in detail.

1.1 Moore and Dennard Scaling

In 1965, Gordon Moore presented the paper [3] which set the historical trend, most commonly known as "Moore's Law", of doubling the number of transistors on a die in every two year (Initially doubling was predicted for every year, but in 1971 it was changed to a 2 year cycle, See Figure1.1a). After 9 years of Moore's Law, Robert H. Dennard co-authored the paper [4], which gave the clear technological insight of Moore's prediction. This phenomenon called Dennard's scaling or MOSFET scaling, which stats: the power density stays constant if voltage and linear dimensions of a transistor are decreased by the same factor. In other words, when scaling a transistor's feature size and voltage, the higher switching frequency is achieved without increasing its power draw. Moore's law gives an effective cost model depending on transistor yield, manufacturing cost and complexity of integrated circuits as shown in Figure 1.1b. The price per transistor was optimized by identifying a favorable balance between transistor count per chip and fabrication yield.

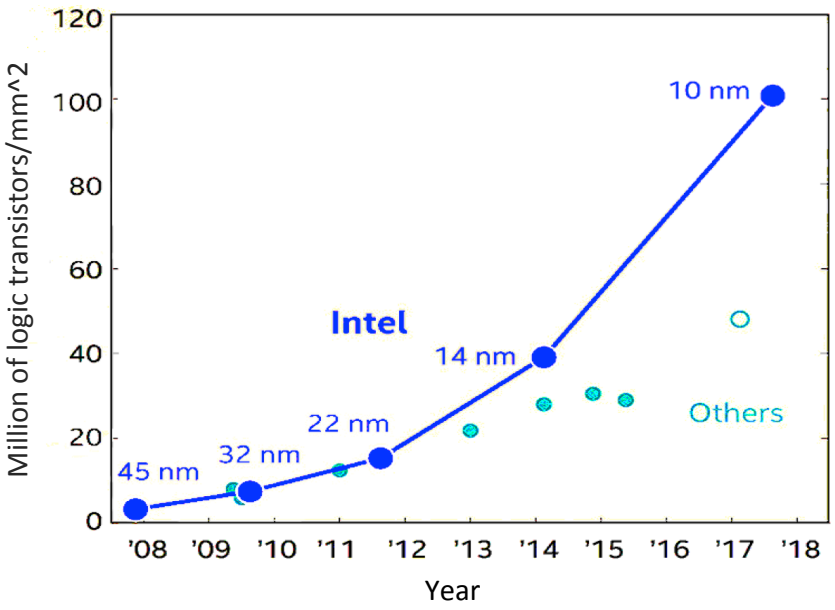
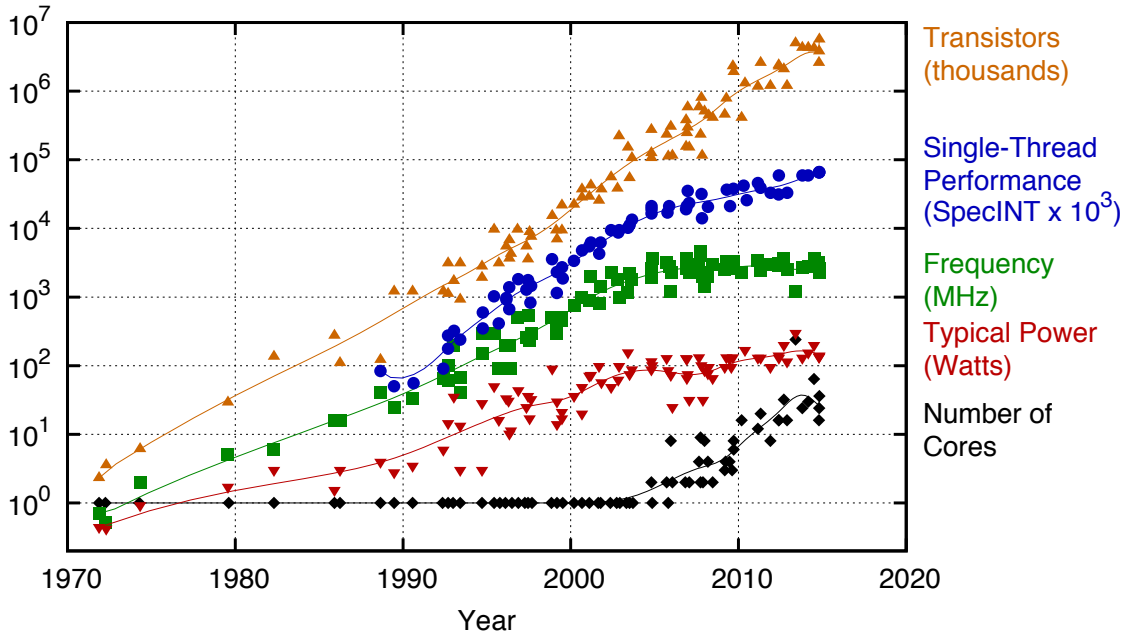


Figure 1.2: Intel trend of transistor density (Source: Intel Corporation[6])

Transistor cost scaling has driven the semiconductor industry for more than half a century. Semiconductor companies were able to reach new technology nodes in rapid succession, not only transforming existing markets but revolutionizing the whole of society. In order to take full advantage of Dennard's scaling which predicts the high speed at lowest power, Intel became the first company to invest billions of dollars to produce faster processors on continuously shrinking technology nodes. The Intel tick-tock model [5] (adapted in 2007) set the roadmap to transition from 45nm to 14nm (soon 10nm) transistor feature sizes. In this model, a new manufacturing process (shrink) is applied to an existing microarchitecture in the first cycle called "tick". Later, in the second "tock" cycle, a new microarchitecture is introduced and fabricated using the previously established manufacturing process. This model enforced Moore's law as shown in Figure 1.2.

For more than four decades, Dennard's scaling was the driving force behind the success of the microprocessor industry. In the middle of last decade, Dennard's scaling encountered a major setback when decreasing transistor feature size did not reduce the power consumption as expected. The continuous reduction in voltage and shrinking of transistor dimensions lead to a substantial source to drain leakage current which increases the static power dissipation significantly. Dennard's scaling stayed valid until 65nm technology, where leakage current and threshold voltage could be ignored, but below 65nm [7] technology the impact of leakage current became more prominent in CMOS design. To keep



(Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp)

Figure 1.3: 40 Years of Microprocessor Trend Data [8]

Moore's law alive, the electronics manufacturing industry turned to other approaches to reduce the power dissipation of a die. Some important trends are listed as follow:

- **Voltage scaling:** Internal voltage of microprocessors is divided into different sections known as power islands. Performing computation in low voltage sections and powering down idle components of a processor are used to reduce the power dissipation. However significant chip area of a processor is used for caches, which cannot be powered down without losing data.

- **Frequency scaling:** Similar to voltage scaling, multiple clock domains are used throughout a single integrated circuit to reduce power dissipation. This frequency scaling reached a barrier (known as "Power Wall") at roughly 4 GHz for the general-purpose processors. As shown in Figure 1.3, since 2006, this barrier is typically not crossed, even though the feature sizes keep on shrinking.

- **Multi-cores:** Multi-core systems are introduced to increase the performance in absence of further frequency scaling by distributing the compute tasks to multiple processor cores. This approach can only yield benefits for compatible workloads (observing a high degree of parallelism). In theory (Amdahl's Law), having infinite cores in a system cannot achieve more than 2x speedup for a task with 50%

parallelism. In practice, multi-core performance is limited by several factors. Most importantly these are: inter-processor communication, shared memory and programming model complexity. Large multi-core systems encounter a phenomenon called dark silicon, meaning substantial parts if a processor has to be powered down to not exceed power and heat limits.

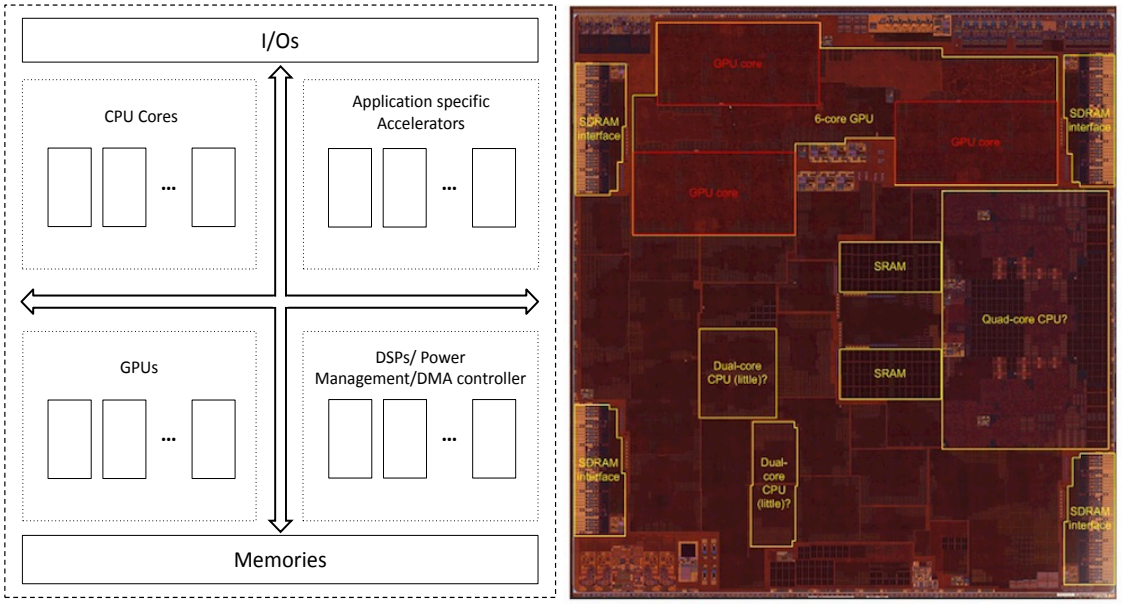
Moore's prediction still keeps going, but soon (as many say [9]) we have to say goodbye to Moore's law and have to look beyond.

1.2 Special Architectures: System on Chip

In the absence of Dennard's scaling, electronic industries started shifting towards specialized hardware architecture to meet the requirements of modern devices. Under the name SoC (System on Chip), this trend led to the integration of more and more off-chip peripherals into a single chip. Allowing higher communication bandwidth, lower production and package cost.

Evolution of SoC design started in the 90s when Integrated Device Manufacturing companies were producing CPU chips and fabless companies were manufacturing GPUs for the PC based market. Meanwhile, other companies were using a sort of on-chip integration of some dedicated hardware module and analog circuitry to meet their product needs. By the late 90s, Qualcomm saw an opportunity of growing cellular market and started developing SoCs by integrating wireless technologies with dedicated hardware ASICs. With the emergence of mobile phones, the SoCs manufacturing business has overtaken the PC market. Soon Nvidia (later Apple) followed the trend and started designing SoC chips using GPU and application processors.

Figure 1.4a shows an abstract overview of modern's SoCs. These are mainly heterogeneous in nature and comprise microprocessors, microcontrollers, Digital Signal Processors (DSPs) and custom application specific accelerators to offload compute-intensive tasks from the central processing cores. An example of modern SoC, A10 die shot used in iPhone 7, is shown in Figure 1.4b. It contains a six core-GPUs, ARM quad cores (2x Hurricane 2.34 GHz + 2x Zephyr). Non-core and non-GPU area of the die indicates many customized application-specific accelerators. Use of specialized processing blocks



(a) Block diagram of SoC example (b) Die shot of apple A10 (iPhone 7) [10]

Figure 1.4: Modern's SoCs

is not limited to the mobile industry, it is also now applied in various other (e.g. Xavier for automotive) industries.

1.3 Hardware/Software Co-Design Challenges

The transition from general purpose CPUs to specialized SoCs comes with new challenges for the design engineers. The design process becomes more laborious task with the rising complexity [2], especially when the system involves multiple processors on the single chip. The designers face great challenges to meet high performance and low power design goals [11]. These system-level engineering challenges are:

- **System-level power estimation:** Detailed power estimation is a tedious and time-consuming task in modern SoC design. The SoC design process is based on multiple iterations; it starts with initial system architecture and gradually it is optimized according to given design constraints. Since the power dissipation is a critical issue next to the performance, it is estimated and analyzed for all components with every

iteration. Usually, the power is simulated late in the chip design flow. Which requires a comprehensive knowledge of low-level synthesis, floor planning and place & route. These power simulations generate large dump files (VCD) exceeding several gigabytes, which are cumbersome to process and take too much designer's time. To speed up power simulations and reduce the design time, a fast system-level power estimation with minimal user interaction is needed.

- **Design space exploration:** Every year the number of components integrated on a SoC are growing, each added component results in the expansion of design space. Which makes it harder for the designer to achieve the desired architectural configuration on time fulfilling the design requirements. The introduction of virtual platforms and rapid prototyping frameworks using SystemC/TLM2.0[12] tried to address issues of the complex SoC designs at system-level. However, the right parameters selection still relies on the designer's experience, which leads to a lengthy development cycle. There is a need for a fully automated DSE tool to find an optimal solution for large design spaces by considering all design constraints in an adequate time.

- **Harware/software partitioning:** Custom accelerators are becoming essential for modern SoCs. This trend poses serious engineering challenges for both hardware and software engineers. Partitioning of hardware and software in early design stages using established design flows is becoming increasingly difficult. Partitioning without evaluating the achievable performance gains can result in poor resource utilization, sub-par performance or costly re-targeting of the architecture in later design stages. The designer can implement hardware prototype using virtual platforms (SystemC/TLM) or hardware emulators at the higher level well before any actual hardware becomes available. However, exploring different hardware/software partitioning using these frameworks remains mostly manual and very tedious task. There is a demand for a higher-level tool for today's complex applications, which can automatically explore possible hardware/software partitioning scenarios for an efficient hardware software co-design.

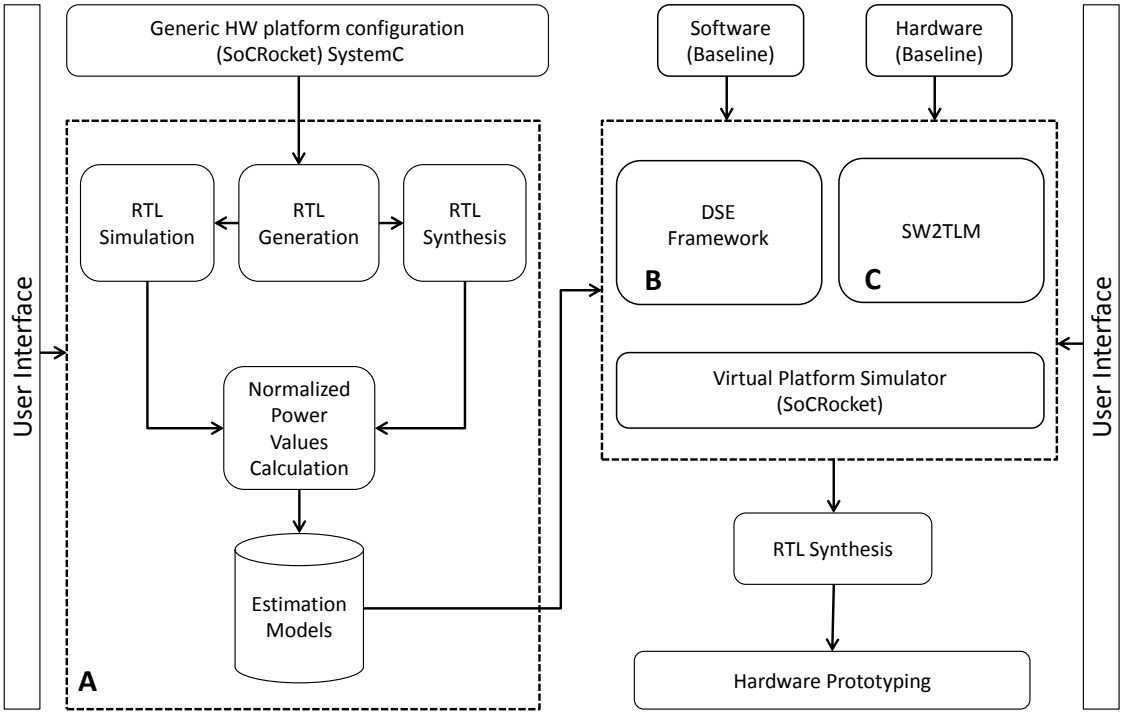


Figure 1.5: Thesis Contributions

1.4 Thesis Contributions

This dissertation presents a series of system-level academic research tools and optimization algorithms for SoC design. The contributions of this work will help designers to explore new hardware architecture using customized application specific accelerator without going into nitty gritty of RTL design. It improves the design space exploration speed to identify energy and performance characteristics for architectural parameters. Figure 1.5 gives an overview of my presented work (marked as **A**, **B** and **C**). Using an automated power estimation Framework (**A**), a designer explores optimized SoC design based on the application and the hardware specification (**B**). While also allowing the easy integration of new custom application specific accelerators with minimum effort and time (C). Finally, energy efficient and optimized SoC design can be brought up for a hardware prototyping on FPGAs.

1.4.1 A: Automated Power Estimation Framework (Chapter 3)

This chapter illustrates the presented Automated Power Modeling Framework which is utilized within the HW/SW SystemC Co-Simulation SoC Validation Platform (SoCRocket[13]). In the first step, the Framework performs RTL synthesis of a SoC design with generic configurations for all components (e.g. CPUs, memories, register files, timers) by using a target technology node library. In the second step, it calculates a power normalized values for all design components by extracting the power estimation from the RTL synthesis reports. Based on these values, the power estimation models are created for each component. It is important to note that the RTL synthesis is done only once and as long as main design components (e.g. processor type) or targeted technology node are not changed, thereby removing the need to repeat RTL synthesis for every power simulation cycle.

These models provide an event based power-monitoring concept, the power is estimated by combining the previously calculated normalized power data with the component configurations and switching information from a system-level simulation. The Framework also provides functionality to update the existing estimation models with new technology node on the fly without creating new models from scratch. When a new accelerator is introduced to the existing SoC architecture, the Framework creates a new power estimation model and combining it with existing ones. The Framework reports instantaneous power dissipation for every component during simulation that can be used in the detection of possible hotspots. At the end of the simulation, a detailed power dissipation summary is reported. In addition, all individual power reports are presented in a GUI representation to be used for later analysis. The approach is demonstrated using a Leon3mp based SoC on a virtual prototype platform (SoCRocket). The contents of this chapter are extended from our PATMOS 2014 publication, A *Lightweight-System-Level Power and Area Estimation Methodology for Application Specific Instruction Set Processors* [14]. I was the lead author of this paper.

1.4.2 B: Automated Design Space Exploration Framework (Chapter 4)

In order to find optimal SoC design parameters in a reasonable amount of time, it is necessary to optimize the design space exploration (DSE) process. The main goal of this chapter to present an automatic and fast DSE extension using the SoCRocket design framework. My contributions include the development of the simulated annealing based optimization techniques and the adaption of an evolutionary computation algorithm for the SoC parameters optimization problem. The presented DSE strategies are integrated into the system-level design platform and evaluated with a range of single and multi-core applications. The design space includes the key parameters e.g. number of processors and the cache configurations. Whereas an energy-delay product (EDP) was chosen as the overall system performance metric. Evaluation results show the presented DSE methodology greatly reduced the number of system-level simulations and DSE overall time by approximately 65%, while maintaining optimal parameter accuracy of 99% as compared to an exhaustive search. The contents of this chapter are extended from our NORCAS 2016 publication, A *Accelerating MPSoC Design Space Exploration Within System-Level Frameworks* [15]. I was the lead author of this paper.

1.4.3 C: SW2TLM (Chapter 5)

In this chapter, a flexible and automated workflow ”SW2TLM” with a high degree of automation to explore different hardware/software partitions using a SystemC/TLM2.0 virtual platform is presented. The SW2TLM, based on sparse configuration, generates a custom TLM hardware accelerator for specific software functions. The timing and power consumption parameters are derived from a high-level synthesis tool-flow. These corresponding parameters are used to create a new power model for the power estimation framework. Subsequently, the hardware accelerator is annotated with characteristics and integrated into the target hardware and software environment. The designer only needs to characterize the part of the software to explore, the achievable speedup and energy savings when offloading it to a specific accelerator are reported by the SW2TLM. These results help a designer in the decision making towards a final hardware/software partitioning, which improves and accelerates the subsequent development process of hardware and

software co-design. The contents of this chapter are extended from our DVCON USA 2017 publication, *Automatic Exploration of Hardware/Software Partitioning* [16]. I was the lead author of this paper.

1.5 Outline of the Thesis

Following this introduction, Chapter 2 reviews the related work in the domain of power estimation modeling, design space exploration frameworks and automatic generation of abstract hardware models. In addition, the principles of the SoC design at the system-level and background of virtual platforms are given. Section 1.4 gave already detailed overview of Chapters 3, 4 and 5. The evaluation of the automated frameworks against several single and multi-core applications is discussed in Chapter 6. In the final Chapter 7, the thesis contributions are summarized and possible further research directions are given.

Chapter 2

Background and Related Work

2.1 Limitation of Electronic System Design (ESD)

With the increasing number of cores, accelerators and complex software applications to be executed on modern SoCs, the design process is getting increasingly complex. Today's smartphones (e.g. iPhone 7) comprise of highly complex multi-core (coupled with many custom accelerators) systems running complex software are examples of such SoC architectures. To keep pace with the growing market of embedded systems, the SoC designers face a difficult design challenge to meet time-to-market constraints. The design complexity has [2] increased since 2003 significantly. This requires a large number of engineers to work in hardware and software teams to deliver products on time. Typically, software team needs to wait for the hardware team to provide a prototype platform. However, research [17] show bigger teams can have a negative effect on the overall of productivity as a significant amount of development time is spent on non-development tasks e.g. for communication between team members or planning. In addition to complex design, limited capabilities of RTL to perform debugging and tracing further increases the pressure of missing production deadlines that can result in costly penalties and missed revenue opportunities.

Traditional ESD design flow is based on RTL, which is a slow process e.g. simulation of SoC booting Linux on RTL models can easily take several days. Creation of testbenches for RTL simulation model is a complicated and time-consuming job. Even if

everything is fine with unit tests, hardware and software co-simulation can uncover the architectural problems, which are not visible until the complete system is tested. Verifying the architecture at this late stage in the design flow results in high costs to re-target the architecture. In order to speed up the SoC design process, an executable specification is needed from early on in the design process. Allowing the software development, debugging and validation to start well before the actual hardware prototype (RTL) becomes available. Suitable abstraction and standard interfaces allow fast simulation and efficient design reuse. In order to enable concurrent hardware and software development the use of virtual platforms is gaining popularity in the modern SoC design flow.

2.1.1 Virtual Platforms

Virtual platform or prototype is system level description of all SoC components supporting software execution. Allowing development of software and drivers significantly earlier in the design. A virtual platform typically emulates processors using an Instruction Set Simulators (ISS), interfaces to all other components (e.g. memories, bus, and peripherals) on the necessary abstraction level [18]. Raising the abstraction level provides a remarkable simulation speedup compare to RTL model simulation [19]. This speedup allows the simulation of a complex system in reasonable time e.g. Linux can boot on system level simulation models within an hour. While hardware prototypes allow much faster execution performance, they lack in other aspects, using virtual platforms the designer gets a deeper insight and control of hardware models that is not the case with HW prototype. With the complete interaction of hardware and software in view, the designer can find bugs, bottlenecks, and structural issues much faster and easier. In addition, virtual platforms provide a seamless implementation of event counters for performance analyses. Through standard interfaces, the re-use of IPs is much easier on the system-level. The virtual platform makes architectural design space exploration (e.g. different memory and caches configurations) faster because there is no need for lengthy RTL synthesis and simulation cycles.

Compared to RTL, timing and accuracy are less strictly defined on the system-level. Allowing to create models directly coupled to a user-case e.g. fast and less accurate models for software developers and high accuracy with lower performance for hardware architects.

Academia and industry are carrying out many research projects for the development of virtual platforms to meet system-level design challenges. Following are some notable virtual platforms based SystemC/TLM.2 [12] standards. Synopsys's Platform Architect [20] is the pioneer of commercial virtual platforms, it was first developed by CoWare, which was acquired by Synopsys in 2010. It is a GUI based tool that provides system-level performance estimation and architectural analysis support. Cadence provide Virtual System Platform [21] is based on an IP catalog (e.g. ARM FastModels [22] and Imperas simulators [23]), it supports co-simulation with the Cadence system development suite (Palladium) and virtual prototype verification with the Cadence verification platform (Incisive). In addition to Synopsys and Cadence, Mentor Graphic is leading EDA (Electronics Design Automation) market. VISTA Virtual Prototyping [24] integrated with Sourcery Analyzer (embedded software analyzer) offers the company's own ESL solution. Arm SoC designer is a GUI-based virtual platform, supports higher abstraction level programmer's view models ARM IPs (FastModel) and cycle-accurate ARM RTL models (Carbon Model Studio). Grammatikakis et al. [25] proposed generic processor model based virtual platform which builds on NoC (network on chip) and integrates a memory access monitor. Lee et al. [26] proposed ARMulator [27] for virtual prototyping of ARM core. Yourst et al. [28] develop a cyclic-accurate simulator of x86-64 micro-architectures and it is integrated with the Xen hypervisor. The gem5 simulator [29] provides a simulation framework covering multiple ISA (ARM, ALPHA, MIPS, Power, SPARC, and x86)), CPU models, detailed memory system and booting Linux on (ARM, ALPHA, and x86), The SoCRocket [13] [30] system-level design framework is developed by Technical University Braunschweig (TUBS) in conjunction with the European Space Agency (ESA) for aerospace applications. The SoCRokect framework is used to demonstrate the contribution of this dissertation and a detail description of the framework will be in Section 2.3.

2.2 System Level Languages and Modeling

In early 1990, the RTL design flow using HDL languages, most prominently VHDL and Verilog, took over from traditional schematic design to cope with growing design complexity. In the same vein, complex SoC design is shifting the paradigm from RTL to System-Level. While raising the design abstraction, HDL languages still provide high

timing accuracy close to the actual hardware, which is useful for accurate power estimation and performance analyses. However, the simulation of high numbers of individual signals and frequents evaluation triggered by the clock signal cause low simulation performance as well as complicated debugging of large systems. The necessity to raise the design and modeling abstraction is a common problem in computer science and engineering. Software developers have been pressured multiple times in the past to transition to new design strategies of higher productivity. So, why not using the same high-level languages as C/C++ or Matlab used in software development to describe hardware architectures on the System-level? Directly applying these languages, however, does not satisfy the specific requirements of system-level hardware description. These languages lack concurrency or timing information. The strict separation of HDL and High-level programming languages lead to a barrier between hardware and software teams, which can have negative consequences on productivity. In order to bridge this gap and combine their advantages, several extensions are proposed in the last few years. These extensions allow simultaneous hardware and software development with speedy simulation through use-case specific simulation models. They incorporate available tools (e.g. GDB and so on), which facilitate the design debugging process. Examples of ESLD languages are: SystemC is an extension of C++, SpecC based on C, and SystemVerilog extension of Verilog. SystemC and SpecC are used in system-level modeling, whereas SystemVerilog is used as verification language.

2.2.1 SystemC

SystemC is a class library for C++ that extends the language with process concurrency, an event-triggered simulation kernel and a notion of time characteristics similar to HDL languages. Its object-oriented style eases the implementation of abstract functional models and design reuse in general. The modules and sub-modules, which are derived from the *sc_module* class, are instantiated similar to VHDL. Structural connectivity between modules is accomplished through interface and exports/ports while processes (computational methods) communicate through channels. SystemC provides a number of primitive channels for synchronized communication modeling between modules and processes, examples of such signals are *sc_signal* and *sc_fifo*. The macros *SC_THREAD* and *SC_METHOD* are used to schedule and synchronize concurrent processes based on events

and sensitivity (analogous to VHDL sensitivity lists). It provides hardware-oriented data types for indeterminate and high-impedance states as well as fixed-point arithmetic e.g. *sc_logic* and sc_logic_vector. While SystemC is compatible with several leading simulation environments, it comes with its own reference simulation kernel allowing standalone simulations. Here, simulation starts from *sc_main* function triggered by a *sc_start* call. The simulation kernel is based on an event-driven algorithm that emulates parallel processing sequentially manner. After the IEEE standardization of SystemC in 2005 [31], it is steadily gaining popularity in industry and academia to model hardware on the System-level. SoCs are typically constructed using IP blocks from different vendors using different communication techniques causing severe compatibility issues when integrated into one system. To address this problem, the Open SystemC Initiative (OSCI, now Accellera Systems Initiative) [12] proposed an interoperability layer for the system-level. In 2008, SystemC was extended with the TLM-2.0 library (illustrated further in the following Section). This extension lays the foundation for modern virtual platforms and virtual prototypes aiming to improve early design explanation, hardware and software co-design, and verification. In addition to TLM-2.0 library, SystemC also provides system-level modeling methodology and technology-specific libraries: AMS-2.0, modeling of embedded analog/mixed-signal systems; CCI, for system-level debug and analysis; SCV, a verification library.

2.2.2 Transaction-level Modeling (TLM)

Transaction-level modeling in SystemC replaces the communication protocols between system components using hardware nets with transaction function calls between higher-level models of those components. It aims at the separation of functionality and communication to improve the interoperability between IP providers [32]. Compared to RTL, modeling on the System-level comes with a higher degree of freedom regarding timing and data handling accuracy. Efforts of the fast few years resulted in the formulation of different coding styles and abstraction levels within TLM [32][33][34][35][19].

In 2008 OSCI TLM working group released TLM-2.0 standard [12] (shown in Figure 2.1 that addresses the taxonomy of abstraction layer for TLM. TLM-2.0 mainly focuses on on-chip memory-mapped bus models and can be extended to network-on-chip (NoC) architectures. TLM-2.0 defines a set of interfaces (API) for different use cases (Software

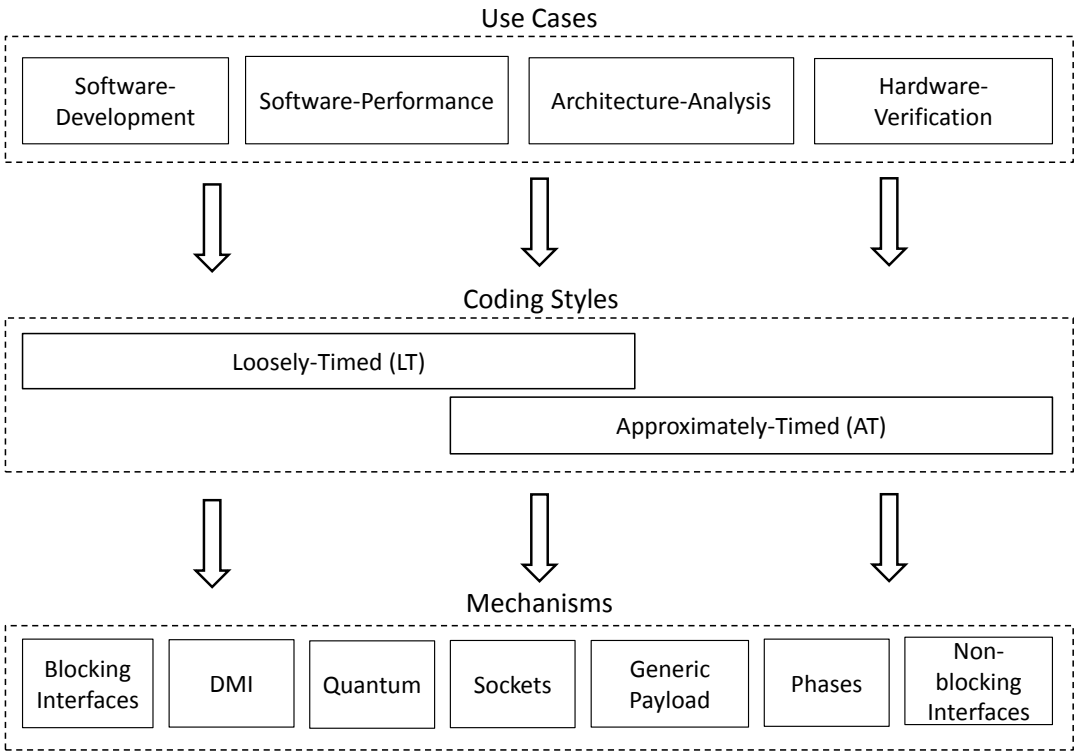
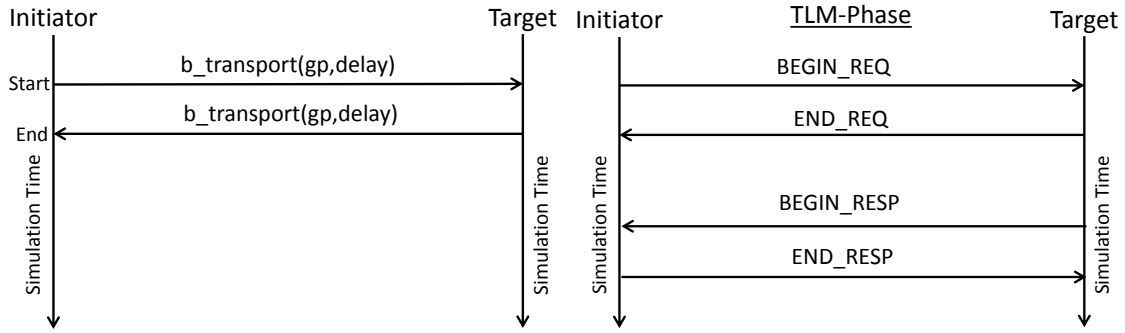


Figure 2.1: Transaction-level modeling (TLM2.0)[12]

development, Software performance, Architectural analysis and many others) to implement low-level transaction-level models with the recommendation of different coding styles along with their implementation mechanism. In the context of the TLM-2.0 standard, a transaction describes a data transfer between an Initiator and Target. The Initiator module starts a new transaction (through an initiator socket) in the direction of the target module (through a target socket). Models can assume the initiator, target or both roles at the same time (e.g. arbiter, bus etc.) by instantiating can connecting the corresponding sockets. The transaction is implemented as a function call transporting a data structure carrying payload and control data. The structure of this transport object is crucial to facilitate interoperability between different vendors IPs.

Loosely-timed (LT) Coding Style: This coding style is applied to user cases, where the timing accuracy has less significance than the simulation performance. The model perspective is also described as Programming View (PV) [19]. A typical use case would be the simulation of a complete boot-up sequence of a complete operating system. LT utilizes different techniques to increase the execution speed of simulations. On important concept is temporal decoupling, allowing processes to run ahead of its simulation time



(a) Blocking (LT) (b) Non-blocking (AT)

Figure 2.2: TL communication

before synchronizing to other components in the system. Another strategy to accelerate frequent memory accesses via TLM sockets it the Direct Memory Interface (DMI), circumventing the simulation of actual transactions by passing a direct memory pointer. As shown in Figure 2.2a, an LT transaction is characterized by two timing points: begin and end times. This is implemented as a single blocking function call (b_transport) with minimal runtime overhead.

Approximately-timed (AT) Coding Style: Aimed at use cases which require higher temporal accuracy e.g. architectural exploration. This coding style corresponds to abstraction level referred as Architect's View (AV)[19] that represents the perspective of the system architect. AT transactions are subdivided into four phase (see Figure 2.2b): Begin_Req, initiator acquire bus and connection is lock(bus is busy); End_Req, target send acceptance and handshake is completed (bus is free); Begin_RESP, target acquires bus and End_Req, initiator sends acknowledgment (bus is free). AT is implemented with non-blocking function calls (nb_transport_fw and nb_transport_bw) in both directions. in the following section, an example of a well-established virtual prototype platform which is built on the System/TLM concepts is described.

2.3 SoCRocket Framework

The SoCRocket [13] system-level design framework was developed by Techincal University Braunschweig (TUBS) in conjunction with the European Space Agency (ESA). It addresses the issues of reusing reliable simulation models at abstract level through a

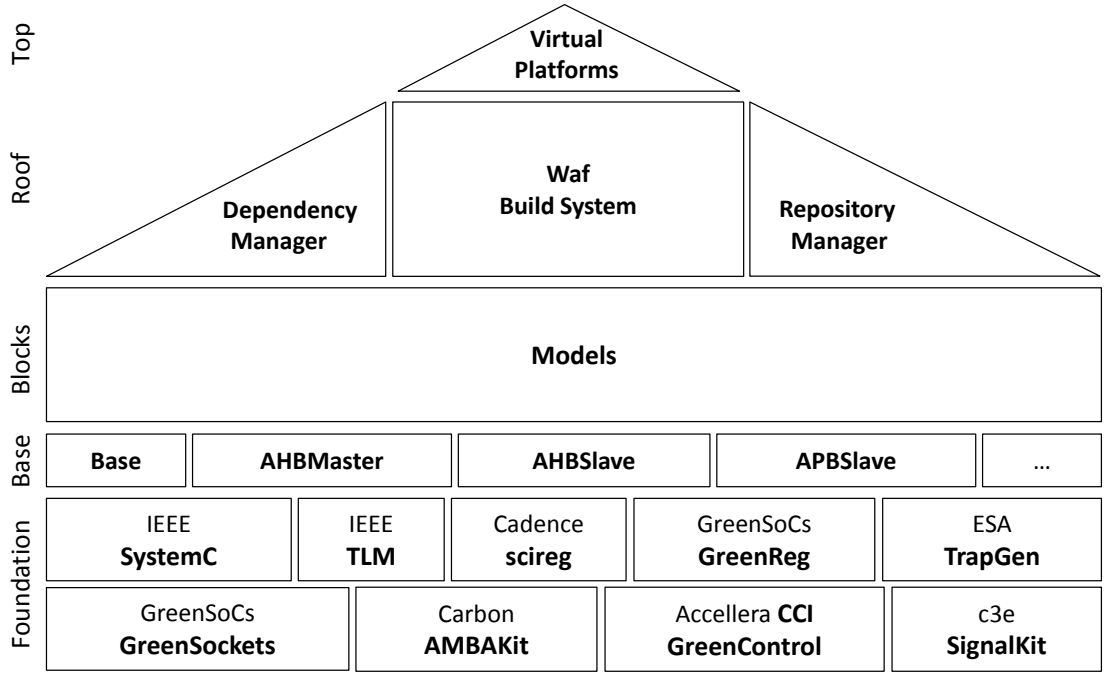


Figure 2.3: SoCRocket Framework Overview

completely open and extensible virtual platform to meet the space industry's special requirements and builds the foundation for space-domain ESL design. The current version of the SoCRocket framework is available as open source [36]. The framework enables design, verification and evaluation of multiprocessor platforms at the system level.

Figure 2.3 describes different layer SoCRocket framework. It shows, the framework's foundation is based on a strong integration of existing standards IEEE 1666 SystemC/TLM2.0[12] with industry partners Cadence and Accellera. One specific aspect of the foundation, which is challenging for all types of virtual platforms, is software visible memory mapped register interface. It is needed by all types of peripheral IP; thus, the next section will illustrate how it is modeled in the SoCRocket framework followed by other key aspects relevant to this work.

2.3.1 Memory Mapped Registers

In absence of a universal standard interface, many IPs providers proved different tools to model registers with SystemC/TLM. One of the approaches to solve this issue is GreenReg provided by GreenSoCs [37]. GreenSoCs is a collaboration platform that focused on the development of open VP infrastructure and offers system-level design solutions including model construction, model to model communication, simulation models and development tools. GreenReg originally developed by Intel's GreenReg framework [38] which simplifies register modeling. The platform offers also special sockets for specific extensions to signal modeling of AMBA, PCIe, and OCP. The SoCRocket register implementation is based on Cadence scireg [39] interface the advantage of the current SoCRocket register implementation over GreenReg is not only the noticeable smaller footprint. It also characterizes by an interface with similar parameters, so it can be applied as a drop-in replacement.

Besides of memory mapped register, a method (SignalKit) is developed for TLM signals implementation. It is based on direct function-calls similar to TLM blocking transports but without payload handling. SignalKit eases the modeling of signal communication by combining SystemC signals with the high simulation speed of TLM-style communication in system-level prototypes of complex embedded systems. Memory mapped register and signal interfacing is closely tied to the software execution. Here SoCRocket supports a wide range of target instruction sets, simulated by custom generated processor models.

2.3.2 TRansactional Automatic Processor Generator (TRAP)

The SocRocket framework is built around the ISS (Instruction Set Simulator) generated by TRAP [40]. TRAP is an open source python-based tool which is available under LGPL license. A designer needs to specify a high level of architecture description which include: the registers counts and size, internal memory, interrupts, internal ports and pipeline stages of the processor. Additionally, the tool also requires instructions behavior specification. Subsequently, the tool will generate the processor ISS based on SystemC/TLM with the support of debugging, profiling and system call emulation. SoCRocket extended these capabilities by exposing a powerful language agnostic API.

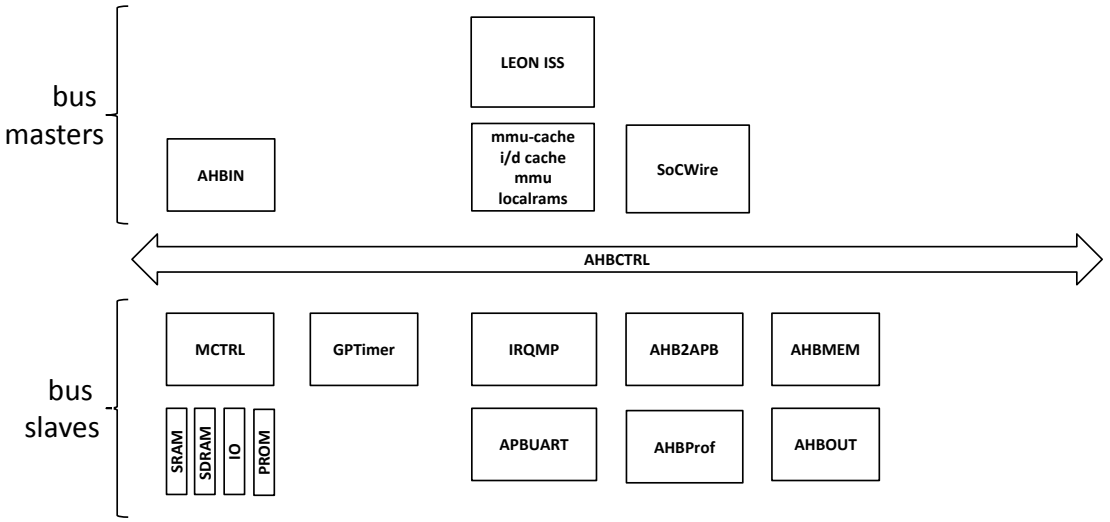


Figure 2.4: An Example of MPSoC System Architecture

2.3.3 Analysis API

The SoCRocket framework provides an analysis API for the implementation of set configuration parameter and performance counters. The analyses API in SoCRocket is based on gs_params from GreenControl provided by GreenSocs. In addition, a new Universal Scripting Interface (USI) [41] was developed in the scope of SoCRocket which complements the original feature set. USI is using the GreenControl SWIG concept. USI enables the designer to delve into the simulation at any time. Similar to GreenControl API, the interface allows register callback function on read and write operations of the model parameters, which makes easier to probe and reconfigure these parameters during runtime. The LII (Language Interface Independent) implementation is one of the unique aspects of the USI, and it can be adapted to any scripting languages like Python, TCL and Ruby. In nutshell, the SoCRoket analysis API provides parameters for model configuration debugging, tracing, runtime-reconfiguration and performance counters setup during the simulation. For baseline IP models and their implementation, standard templates and libraries included in SoCRocket are presented next.

2.3.4 Models and Templates

The SoCRocket framework models are designed from Cobham Gaisler GRLIB [42] which is an open source library and available under GNU GPL license. Figure 2.4 shows the implemented models within the SoCRocket core library. The simulation models (also shown in 2.3 on the Base and Blocks layer) are developed in SystemC and build on the OSCI TLM2.0 standard. The current implementation of the models is in LT, AT and cycle timed which enables to create mixed-abstraction and timing variants depending on the required use case. The LEON3/SparcV8 instruction set simulator is the central component of the SoCRocket platform. The LEON3 processor can be simulated as a stand-alone model as well as a complete SoC with several peripherals. The multi-core SoC has N number of LEON3MP processors embedded on shared a single AMBA High-Performance Bus (AHB) which has plug & play and snooping support. Each of the processors has its own private instruction cache (I) and data cache (D), it can also include local memory management units (sparc MMU) and local rams. A memory controller connected to CPU through shared AHB bus, It has support for four different kinds of memories: SRAM, SDRAM, IO and ROM. SDRAM and PROM are provided as default memory configuration. To complete the system some additional IP like an AHB2APB bridge, and AHB memory and an APB UART are provided. The configuration registers of the MCTRL, the multi-processor interrupt controller and the general-purpose timer are connected through the APB bridge. Quick implementation of new IP is supported by bus access template (AHBMaster/AHBSlave) applied according to the IP's role. The AHBIN, AHBOUT and AHBProf components have no correspondence models in the GRLIB but have been proven useful for simulation control and testing. Moving to the top of the Figure 2.3, a versatile infrastructure setting up the environment as well as automation of compilation and test is provided.

2.3.5 Build System and Execution

The build system of the SoCRocket framework is written in python based waf which is an automation tool. It automatically sets up the framework, checking and fulfilling third-party dependencies and performing initial build configurations. To structure collaborative work into different directions, a repository manager is included, allowing separation of

the base library and new IP development. On top of above-mentioned infrastructure, a self-contained executable VP is compiled. After getting to know the SoCRocket, the next section will elaborate the practical design flow for novel embedded architectures.

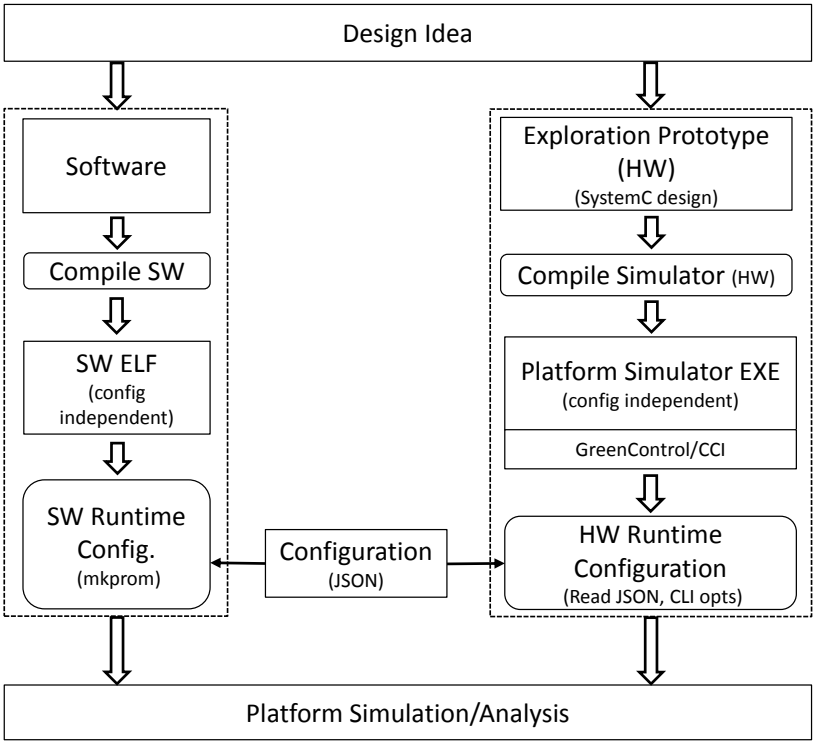


Figure 2.5: SoCRocket: Design Flow

2.4 Design Flow of SoCRocket

The design flow of SoCRocket shown in Figure 2.5, it describes the path from a design specification or reference software to a virtual prototype. First, the designer needs to split the design into a hardware and into software part. This hardware part is considered as an exploration prototype (EP), it is a SystemC design template containing the system parameters to drive the exploration. The modeling library (as described in the previous section) contains some predefined EPs, particularly for standard LEON systems so that the designer does not need to start from scratch.

```
1 {
2   "conf": {
3     "system": {
4       "clk": 20,
5       "ncpu": 2
6     }
7   }
8 }
```

Listing 2.1: JSON file example of configuration parameters

These template EPs can be easily extended or modified. The software which is written in C/C++ code is compiled using SPARC compiler and executable binaries of the software are supplied to the simulator with a boot code file (PROM image). The pre-compiled boot code is used to initialize the run-time system and the peripherals.

The hardware configuration parameters can be extracted from the EP, along with default values, value ranges, and descriptions. These parameter values are later used for the performance analysis. The hardware parameters that will be subject to exploration can be supplied to EP via JSON (JavaScript Object Notation) file during simulation run-time. Listing 2.1 shows an example of JSON file, it contains two re-configurable systems parameters: the system clock (line 4) and the number of CPUs (line 5).

```
1 {
2       gs::gs_param<unsigned int> p_system_ncpu("ncpu", 1, p_system);
3       gs::gs_param<unsigned int> p_system_clock("clk", 10.0, p_system);
4        ....
5     for( uint32_t i=0; i< p_system_ncpu; i++)
6     {
7             Leon3 *leon3 = new Leon3( ... );
8           // Set clock
9           leon3->set_clk(p_system_clock, SC_NS);
10           ...
11      }
```

Listing 2.2: Initialization of number of CPUs

Listing 2.2 demonstrates reconfiguration procedure. Here the configuration parameters (line 2 and 3) are initialized from the underlying GreenSoCs/GreenControl API at run-time. The EP creates and binds processors components in a loop iterating from 0 to ncpu (lines 5-11). All parameters of SoCRocket components are specified as GreenSocs parameters (gs_params), therefore can be modified easily without re-compiling of application software and exploration prototype.

Execution statistics are collected through performance counters of the models during the simulation. Which are then written in the form of log [43] or waveform files at the end of the simulation. These results are carefully analyzed by the designer if the performance goals of the design are not achieved then the configuration or partitioning must be optimized.

As it can be clearly observed from the design flow, designer intervention is required at many stages e.g defining the hardware and software partitioning or design space exploration process for selecting rights parameter configurations. This poses a huge challenge for the system designer and makes hard to fully leverage the speedy design through virtual prototyping. This work targets the automation of hardware and software co-design as well as speeding up the process of finding the optimal design parameter configuration. Moreover, light weighted and flexible power models are designed and added to the framework. Building on top of the presented virtual prototype platform background, the next section explores existing approaches related to the contribution of this work in the scientific literature.

2.5 Related Work

Modern virtual platforms observe a raised level of abstraction through the application of high-level programming languages and methodologies such as SystemC and Transaction-Level Modeling. The framework presented in this work utilizes these techniques to improve system-level architecture exploration for multi-processor systems designs. In recent years, academia and industry exploring system-level design intensively to short the gap between hardware and software production. In the following, the latest research activities in the domain of system-level power modeling, micro-architectural design exploration and hardware software co-design are reviewed.

2.5.1 Power Modeling

With the increasing clock frequency, the early power estimation of SoC is becoming more important. These estimates enable designers to find an optimal architectural configuration with the best power and performance trade-off. There are several considered approaches to system-level power modeling. The traditional method of power modeling is called spreadsheets approach [44] where the power values are obtained from the data sheets of individual SoC components. This approach is mainly for static power estimation. It doesn't take switching activity of the running application into account; therefore, the application dependent part of power dissipation cannot be estimated by only using the spreadsheets. In state machine-based power modeling, firstly the gate-level power consumption of processor and other components in steady states and state transitions is computed. Later, the power estimates of each state are calculated through simulation-based or analytically models [45][46]. Cycle-accurate modeling is a low-level modeling approach close to RTL representation, where cycle accurate timing is defined and every clock edge is taken into account. Cycle-accurate power modeling uses the micro-architecture (data pipeline, caches etc.) models and the bit accuracy of data transfer to obtain the power estimates. SimplePower [47] and Wattch [48] are examples of cycle-accurate power estimation tools. This power estimation approach observes high accuracy compared to RTL power simulation and is applicable to different architectures. However, this cycle accuracy results in a slower simulation which is a bottleneck in architectural exploration. Specific architecture templates related power estimation is explored in work of [49], in which the cost estimation is done for transport triggered architecture (TTA) template of the Move framework [50].

Negri et al. [51] proposed a work-target power simulation for wireless communication protocols based on SateC. This approach is not suitable for other general SoC design flow. The approach of Ahuja et al. [52] works only for specific operation modes. Basically, they write assertions in the design flow for power estimation. Bogliolo et al. [53] introduced gate-level power estimation using regression-based power macro-models, their approach is limited to structural RTL macro of which leaf components are combinational logic blocks and registers. UML based model-driven approaches for early power design are presented in [54][55][56][57][58]. These approaches target only power aware code generation using system-level power estimation models.

This work investigates the TLM power modeling, the corresponding research field is very active and attracts continuous contributions. Onnebrink et al. [59] focused on accelerating the simulation performance by reducing model accuracy. Techniques like DMI, based in the LT coding style, are used to circumvent simulation overhead caused by the memory access hierarchy. This induced error is acceptable for regular architecture; however, this approach is not applicable to heterogeneous SoCs which observed complex communication patterns. Grüttner et al. [60] presented a rapid prototyping framework for heterogeneous SoC. It leverages high simulation performance by executing target software natively on the host system. Comparable to the simulation of hardware modules, the software is annotated with power and timing information to achieve realistic exploration results. Here the designer must evaluate the trade-off between performance and accuracy. Bouhadiba et al. [61] uses SytemsC/TLM stimuli to feed external commercial tool ACEplorer (since 2015, it is part of Intel Docea solutions [62]) to estimate power. The power model through the least-mean-square method by a set of 50 configurations is proposed in [63], which only targets the ASIC processors models. Caldari et al. [64] and Bansal et al. [65] presented frameworks which use power-models of the components at the system level stage. Latter approach selects the most suitable power-models for the individual components relying on efficiency and accuracy tradeoff. Similarly [66] employs this trade-off by exploiting co-simulation techniques for power estimation. They propose techniques to speed-up the co-simulation process. Negri et al. proposed McPAT framework [67], based on CACTI [68] cache model toolset, for the area, power and time estimation for the multicore system. To feed the data into power estimation, the framework uses an XML interface between the simulator and the power models. XML Interface is cumbersome compared to the human-readable JSON interface approach which simplifies the mapping (reading and writing) of objects without taking care of the underlying modeling languages. In [69][70][71][72] activity-based power models are presented which convert RTL design of power estimates and manually annotated them to TLM models.

In contrast to previous work, this work targets TLM power modeling that includes a dynamic creation of power models which allows technology node re-configurations at runtime during system simulation. Automatic plug and play of new models and hotspot detection feature through instantaneous power reporting of the presented power estimation framework give it a distinctive edge over the preceding work.

2.5.2 SoC Hardware-Parameters Exploration

Design space exploration is a highly time demanding job during the system level design of MPSoC. An efficient DSE methodology and its integration into a rapid design development framework play an important role in the fast design space exploration. In [73], Mohanty et al. presented the hierarchical DSE methodology which is implemented by the integration of symbolic design space search and cost estimation tools. An ASIP design space is explored in the work of [74][75]. Many methods based on analytic models are proposed to evaluate the design points by analyzing the performance/cost of each point [76][77][78].

To increase both the speed and mapping of applications with complex architecture requirements, the system level simulation based models are used to optimize the design space coverage of single/multi-objectives, as presented in [73][79][80]. The system level design space is usually exhaustively searched to achieve the optimal design-goal [13][81][82][83], which is very tedious and time-consuming work.

Mahapatra et al. [84] proposed method called FSA to target HLS architectural exploration using CyberWorkBench [85] HLS tool. This method is based on a decision tree which is generated from the training data. In order to generate the decision tree, FSA needs to simulate half of the design space which limits its applicability to a larger design space. Similar work for integrating the CyberWorkBench tool into design exploration is presented in [86]. This approach is based on adaptive of simulated annealer, which explores the design space for the specified area and latency attribute in a pre-defined number of simulations. Karanjkar et al. [87] presented simulated-based optimization technique for multi-core exploration. This technique first converts a discrete space problem to continuous search space then employing ergodic interpolation to find the optimal solution. The quality of continuous space solution suffers from rounding error when it is converted back to discrete space. A genetic algorithm-based tool called UTNoC is presented in [88]. This tool generates area and performance optimized NoC topologies. It does not support NoC architectural explorations. Targeting uses case of embedded control units (ECU) in the automotive sector, [89] presented design space exploration for MPSoC by introducing an additional level of architectural variation into exploration. This approach allows SoC architecture optimization based on different ECU types. Despite the abstraction, the speed of exploration is limited by the size of the design space. Mediouni at el.

[90] proposed genetic and variable neighborhood algorithm-based exploration of optimal configuration for FPGA based SoC in terms of area and execution time. In this method as a first step, the application is divided into different tasks for target software and hardware partitioning then exploration starts for an optimal configuration. This process is done manually which is very slow and extra overhead to design exploration. Papamichael at. [91] proposed Nautilus, an extension of GA for IP parameters tuning. This approach is evaluated on FPGA designed IP for the area, frequency and other IP specific parameters. However, Nautilus requires the designer pre-experience in the exploration for faster results, as well as the evaluation framework requires FPGA IP synthesis with selected parameters through the process which affects the exploration speed. In a similar direction, a promising work is done by researchers at IBM and Columbia University New York. They developed a tool name SynTunSys for automating synthesis parameters tuning process, which is deployed in the design phase of e IBM z13 processor chip. Industry and academia provide several design frameworks [92],[93][94][95][96][97] for the exploring the design space of MPSoCs with above mentioned optimization techniques. The following paragraph outlines notable DSE frameworks.

Jia et al. [98] presented a DSE framework called NASA (Non-Ad-hoc Search Algorithm). It is implemented in C++, it allows integration of different simulation tools and can incorporate different design space search strategies. An architectural generator is also integrated into the framework. The framework's design is based on a modular approach consisting of several modules including integrated an architectural generator module. These modules use three XML interfaces to communicate with each other. Although, this framework can integrate different search methods from other DSE framework it lacks its own search methodology other than a GA implementation. Durillo et al. [99] developed the jMetal framework in object-oriented Java. It utilized a genetic algorithm along with other metaheuristics for multi-objective optimization. The framework has a platform independent API which can be easily utilized by the designer in the exploration process. However, Java-based implementations observe lower execution performance and higher memory requirements, which impacts the overall framework performance. The DSE framework developed in MULTICUBE project [100] is focusing on micro-architectural exploration for multi and many core SoCs. The framework infrastructure consists of two tools: MULTICUBE Explore [101], an open source optimization tool and modeFRONTIER [102], a proprietary multi-objective optimization tool. It can be reconfigured via

an XML interface to any configurable simulator and it targets power and performance trade-off for SoC design. An optimal architectural configuration is searched by using different simulation and analytic-based heuristics. The EU funded project COMPLEX [103] also focuses on the analysis of large MPSoCs design space regarding overall system energy consumption estimation. The project's outcome is a design methodology combining optimization techniques of energy consumption with the virtual prototyping [104]. VP under test is generated from UML and C/C++ description and processor modeling is done through ARMulator [27] and SimpleScalar [105]. Simulation traces of energy consumption over time is used as an input for the DSE process. The DSE can be controlled in an automatic and semi-automatic way where designer experience plays a significant part in the design exploration.

Above mentioned DSE framework either need human intervention or require very lengthy scripting for automation. In this work, several DSE strategies (metaheuristics) considering single/multi-objectives are pretested, which significantly reduce the DSE time as compared to exhaustive search technique. The DSE methodology is integrated into a state-of-art virtual platform framework called SoCRocket and it is validated with the help of several single/multi-processor benchmarks and applications.

In contrast to previous works, the presented framework is fully automated, and it does not need human feedback. It includes a scriptable reporting and logging framework [43], which can greatly reduce the amount of data generated during DSE process. This increases framework and designer efficiency. Furthermore, It uses a universal scripting interface (USI) [41] to generate VHDL code from SoCRocket platform configurations. This enables, to get power values from an RTL simulation through power estimation models (presented in this work) and automatically feedback those values into the DSE module.

2.5.3 SoC Hardware/Software Partitioning

The tool-supported translation of software sections to TLM models is investigated in academia as well with commercial tools. Especially computationally demanding fields steer towards higher specialization of hardware and software architecture. Computer vision is a typical domain in need for acceleration to analyze significant amounts of data at

a high rate. Mefenza et al. [106] presented a rapid prototyping framework to evaluate different partitioning and mapping strategies based on an OpenCV computer vision library. The framework spans different levels of abstraction, performing high-level analysis using SystemC/TLM [12] hardware models as well as validation of actual RTL models on an FPGA target platform. Other high-level exploration approaches target specific design goals instead of concentrating on functional verification. Zuo et al. [107] present a framework generating SystemC models for exploration purposes towards low- power architectures. Based on a subset of C/C++ (specifically affine program regions) the framework computes power and performance characteristics to subsequently generate realistic SystemC models of a corresponding accelerator component. The generic nature of the generated architecture deliberately targets early design stages. Mück et al. [108] present an approach to integrate hardware and software components into a virtual platform based co-simulation environment. It is demonstrated how to utilize metaprogramming techniques common to OOP to facilitate transparent communication between partitions of functionality implemented in hardware or software. In addition, a NoC interconnect is provided to facilitate a higher number of actively communicating hardware accelerators.

Typically, these DSE frameworks aim to identify the most suitable architecture without exposing the complexity and effort of actual hardware development to the user. This became possible by the growing number of high-level synthesis tools, generating RTL hardware models based on available (high-level) software code. A recent survey of different commercial and academic high-level synthesis tools can be found in Nane et al. [109] These tools can be specifically targeted to a domain as well as general purpose solutions. Addressing the advantages and disadvantages of different approaches of HLS is beyond the purpose of this work and following only a few tools are exemplarily presented. One example is CatapultC [110] (product of Calypso Design Systems; used in Mück et. al.), it generates synthesizable cycle accurate code based on C/C++ code or already available SystemC models. Other solutions also provide functionality beyond synthesis; Legup [111] demonstrates how profiling information can be directly utilized to decide on a hardware/software partitioning (i.e. offloading compute-intensive software sections to an FPGA fabric). By skipping the DSE step on a higher abstraction level this either requires time- consuming simulation runs on RTL or the availability of the target hardware. To address high-level DSE, the generated RTL designs themselves are typically not directly utilized as their simulation performance is not sufficient. However,

they are a good source of information regarding the expected performance and power consumption of the actual hardware accelerator. Relaying those parameters back into the high-level models allows the simulation environment to deliver accurate estimations used in the DSE. The solution presented in this work is unique as it integrates seamlessly into a state-of-the-art SystemC/TLM2.0 virtual platform without exposing all its complexity to the designer exploring different hardware/software partitioning choices.

2.6 Summary

This chapter described the background and limitations of ESD. Virtual prototyping was presented as a solution to these limitations. The developments of virtual platforms were highlighted and important VPs from academia and industry were discussed. After that, a background of system level languages and modeling was given. Current trends of system-level modeling with SystemC and TLM2.0 were thoroughly illustrated. An overview of SoCRocket Framework, a state of art VP, was presented. Finally, with the reference to this dissertation, the current research activities in system-level power modeling, micro-architectural design space exploration and hardware/software were analyzed.

Chapter 3

High-level Power Modeling and Analysis

The overall power minimization potential at ESL (behavioral and architectural) is about 80% [112], an efficient power modeling can guide a SoC designer to exploit this potential by using low power techniques (e.g. clock, power gating), suitable memory configurations and the right selection of hardware and software partitioning. The total power consumption of an electronic design can be divided into two major components: static power and dynamic power (see Equation 3.1).

$$P_{Total} = P_{Static} + P_{Dynamic} \tag{3.1}$$

Static power is consumed by transistor regardless of switching/ state changes; it is caused by a leakage leading to the name leakage component of the total dissipated power.

$$I_{Leakage} = I_{Sub-threshold} + I_{Gate-oxid} \tag{3.2}$$

Equation 3.2 shows the leakage current mainly consist of sub-threshold and gate-oxide leakage. For an NMOS (n-channel metal-oxide semiconductor) transistor; sub-threshold leakage is weak inversion current that is flowing from drain to source and the gate-oxide current is leaking through gate oxide insulation. The leakage current depends on the transistor feature sizes, with deeper submicronic technologies ($< 90nm$), the leakage current becomes more dominant which adversely affects the overall power dissipation. This

makes static power a highly important characteristic, that needs to be modeled throughout the design flow. The static power is more or less independent application running on the system and system clock frequency. It is directly proportional to silicon area of the chip and strongly depends upon semiconductor technology.

$$P_{Dynamic} = P_{Internal} + P_{Switching} \tag{3.3}$$

The total dynamic power is the sum of internal power and switching power (see Equation 3.3). The internal power consumption is caused by short current (VDD to ground) within the cell. Similar to the static power, the internal power is also independent of the application execution, but it has a linear dependence on the chip-area and clock frequency. Whereas, the switching power is the result of charging and discharging of output capacitive loads of the cell. It depends on the rate of change between logic states (logic activity while executing an application) and clock rate. Power estimation modeling at system-level allows the designer to incorporate most commonly used low power techniques such as clock-gating and power gating early in the design flow. Clock-gating is used to reduce the dynamic power and the power gating is used for static power reduction. However, these techniques require defining the semantics of low- power intend for system-level architectures, which was not in the focus of this work and it could be investigated in the future work.

3.1 Power Estimation Methodology

Usually, power analysis is carried out on various stages of the design flow, the accuracy of power analysis is increasing from high-level (ESL) to low-level (gate level) of the design flow. Whereas, the speed of power simulation decreases from top to bottom of the design. In industry, power analyses are typically performed on RTL models by using synthesis reports of EDA power simulation tools. Power simulation of an RTL design is a tedious job, it requires a design written in VHDL/Verilog or another hardware description language, it lacks simulation performance and produces a large simulation trace that is difficult to process. Final power estimates of designs are obtained from these power simulation reports. This approach is not suitable for system-level design, where architectural exploration requires many design iterations.

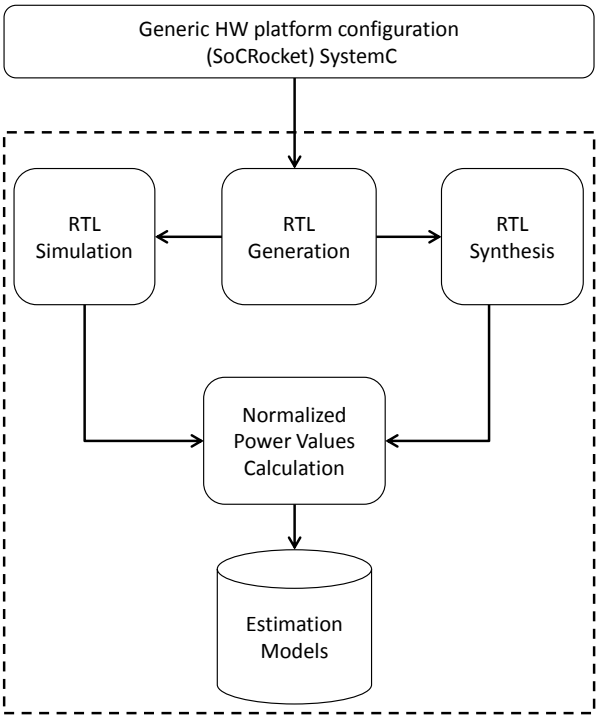


Figure 3.1: Power estimation design flow

System-level design is an event-driven design; an event is generated whenever a system's component is accessed i.e. read or written. So, system-level power estimation should be based on the event-triggered power-monitoring concept. Figure 3.1 shows the presented methodology, a templated design configuration either in the form of RTL or ESL (when written in SystemC then a corresponding RTL of the design is generated) is synthesized using ASIC technology. The reported power consumption parameters are extracted and used in the system-level estimation models. These models enable reuse of the RTL power parameters on ESL, which allows a system-level designer to fast gauge for optimal architectural parameters with reasonable power estimation accuracy. In the following section, the presented methodology is discussed in detail.

3.1.1 RTL Generation

Figure 3.2 shows a workflow to generate VHDL from SystemC, the SoCRocket platform metadata (component name, type, size etc.) is extracted and stored in a JSON configuration file. This JSON file together with a template (VHDL files with generic configuration)

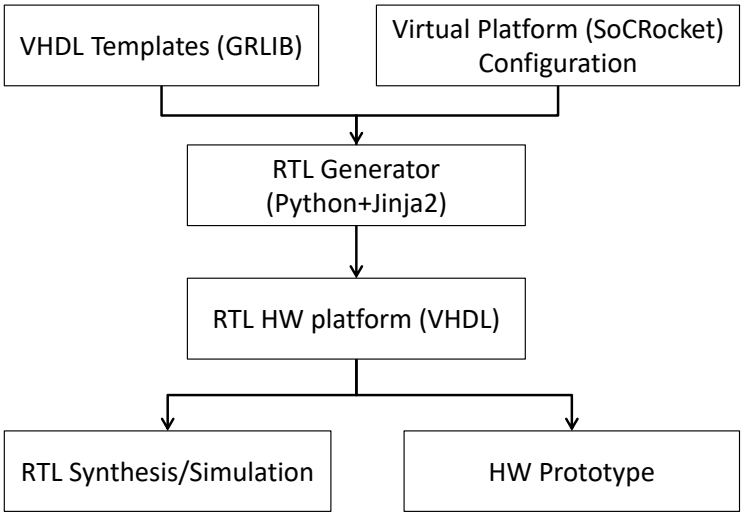


Figure 3.2: Workflow from SystemC to VHDL

from GRLIB is passed to RTL generator, which utilizes a Jinja2 template engine [113] and data type conversion python scripts. The workflow of RTL generation is as follows: GRLIB VHDL template modules are compared to SoCRocket modules, all modules those are not present in the SoCRocket are deactivated. In case FPGA prototype is required for verification and power simulation, the specific modules might need to be reactivated (if there any errors in simulation). Translation of SoCRocket module nomenclature and conversion of variable data type Hexadecimal/ Boolean to VHDL conform notation of the particular numeral system is performed by using the implemented python functions (set_var and create_generic_map). A separate repository of VHDL files is created, which can be used for synthesis and simulation. The complete overview of the workflow can be found in [114].

3.1.2 Normalized Power Values

To make RTL power estimation results usable on the System-level, they must be transferred to a higher abstraction level. This step is called normalization and allows the annotation of power consumption values to abstract transaction-based models. The power estimation models are based on normalized values, these normalized values are derived from the reference power estimates of one-time conventional RTL power simulation, reported after place & route of a generic design configuration of system components. The

normalization calculation procedure varies for static power and dynamic power. As mentioned before the static power component scales with silicon area and it is independent of clock rate. The internal component of dynamic power is dependent on the area as well as the clock frequency. Given the reference values of static and internal power of a component, the normalized values can be calculated by Equations 3.4 and 3.5 respectively.

$$P_{Static\_norm} = \frac{P_{Static\_ref}}{D} \tag{3.4}$$

$$P_{Internal\_norm} = \frac{P_{Internal\_ref}}{D \times f_{ref}} \tag{3.5}$$

Where D in the denominator of Equations is called normalization parameter. It varies for different components. For instance, in the case of memories (e.g. SRAM, ROM) it is specified in total storage capacity in the number of bits. The details of this factor D for different components are given in Table 3.1. f_ref is the reference clock from the RTL synthesis simulation, this is utilized in the normalized values calculation, which allows a system-level designer to vary the clock frequency in the design. The switching part (application dependent) of dynamic power represents the average consumption of energy over time and it is not suitable for discrete event-based system-level simulation. The normalized value of switching energy per access is used for switching power modeling at system-level. It is calculated by Equation 3.6.

$$E_{Switching\_norm_{access}} = \frac{P_{Switching\_ref}}{B \times t_{rate} \times f_{ref}} \tag{3.6}$$

Where normalization parameter B depends on the number of executions or number of accesses (read/write), the detail of B for different groups of components is presented in Table 3.1, t_{rate} (activity toggle rate) and f_ref (reference clock) are used in the RTL power simulation.

3.1.3 Power Estimation Models

To leverage the reuse of RTL power simulation estimations at system-level, the normalized power values of the components need to be integrated into suitable power estimation models. At system-level abstract events representing complete transitions (read/write)

Component	**Categories**	D	B
AHBCTRL	a	$n\_masters + n\_slaves$	$n\_masters + n\_slaves$
LEON3	b	1*	1
AHBMEM	c	n_bits	$n\_bits * width\_bits$
GPTimer	d	n_timers	n_timers

* Same as the reference value.

Table 3.1: SoCRocket Components (based on GRLIB) Normalization Factors

The system components, depending on their configuration, structure and functionally are divided into four major categories: **a**, data routing network (Bus Controller); **b**, computational logic (Processor, Accelerator); **c**, memory-based components (e.g. SRAM, ROM); and **d**, some miscellaneous components (GPTimer, IRQMP) which are not fit into first three, every component is modeled separately.

compared to RTL clock driven activations, these events necessitate an event-triggered power-monitoring concept. Whenever a component is accessed during simulation an event is recorded, a counter is maintained for the event. This information is used in the component power estimation model to calculate instantaneous/total power consumption during or at the end of the SystemC simulation. The de-normalization for a given power input parameter is performed in the estimation models by using normalization factors B and D (see Table 3.1). Equations 3.7 and (3.8) show the estimation of static and internal power consumption respectively.

$$P_{Static\_est} = P_{Static\_norm} \times D \tag{3.7}$$

$$P_{Internal\_est} = P_{Internal\_norm} \times D \times f_{design} \tag{3.8}$$

Where f_{design} is clock frequency of design. Switching energy is de-normalized in Equation 3.9 for individual accesses, then this de-normalized energy per access is multiplied by the number of read or write access events (or the number of instructions, in case of CPU) occurred during simulation to estimate the switching power as shown in Equation 3.10.

$$E_{Switching\_est_{access}} = E_{Switching\_norm_{access}} \times B \tag{3.9}$$

$$P_{Switching\_est} = \frac{E_{Switching\_est_{access}} \times n\_accesses}{simtime} \tag{3.10}$$

Where in Equation 3.10, $simtime$ is selected by the user to define the granularity of instantaneous power reporting. If it is not given, the whole simulation time is used to calculate the average power consumption of the design as shown in the Equation 3.11.

$$simtime = \begin{cases} \Delta t & : t_{start} < \Delta t < t_{end}, if\ specified \\ t_{end} & : if\ not\ specified \end{cases} \tag{3.11}$$

3.1.3.1 Example of Instruction Cache Power Modeling:

As an example, the power modeling of the module instruction cache $(icache)$ is discussed below. The power modeling of the $icache$ is divided into three parts: controller $(icctrl)$, tag rams $(itram)$ and cache ram $(icram)$. The total power of $icache$ is calculated in Equation3.12.

$$P_{total}^{icache} = P_{total}^{icctrl} + P_{total}^{itram} + P_{total}^{icram} \tag{3.12}$$

The static power the $icctrl$ are considered approximately constant. Removing the need to de-normalization for $icctrl$ static power estimation $((P_{Static\_est}^{icctrl}) = (P_{Static\_norm}^{icctrl} \times))$ and estimated internal power of $icctrl$ is also nearly the same as normalized $(P_{Internal\_est}^{icctrl}) = (P_{Internal\_norm}^{icctrl}) * f_{design})$. Whereas, the static and internal power of the rams ($itram$ and $icram$) are linearly depended on the capacity, additionally the internal power is also depended on the clock frequency. The estimations of static and internal power of $(icache)$ are given in Equations3.13 and 3.14.

$$P_{Static\_est}^{icache} = P_{Static\_est}^{icctrl} + N_{sets} \times (P_{Static\_norm}^{itram} \times N_{tbits} + P_{Static\_norm}^{icram} \times N_{icbits}) \tag{3.13}$$

$$\begin{aligned} P_{Internal\_est}^{icache} = f_{design} \times (P_{Internal\_est}^{icctrl} + N_{sets} \times \\ (P_{Internal\_norm}^{itram} \times N_{tbits} + P_{Internal\_norm}^{icram} \times N_{icbits})) \end{aligned} \tag{3.14}$$

Where:

N_{sets}: is the number cache banks

N_{tbits}: is the size of instruction tag ram in bits

N_{icbits}: is the size instruction cache ram in bits

f_{design}: is the clock frequency of design

For Switching power, first, the switching energy per access of $itram$ and$icram$ is calculated (See Equation 3.15 and 3.16), for simplicity the read and write energy is considered as equal. After that, the switching power of $icache$ is estimated with respect to simulation time as shown in Equation 3.17. The complete estimation models of all components developed within this work can be seen in [14] and [36].

$$E^{itram}_{Switching\_est_{read}} = E^{itram}_{Switching\_est_{write}} = E^{itram}_{Switching\_norm_{access}} \times Width_{tbits} \times N_{tbits} \tag{3.15}$$

$$E^{icram}_{Switching\_est_{read}} = E^{icram}_{Switching\_est_{write}} = E^{icram}_{Switching\_norm_{access}} \times Width_{icbits} \times N_{icbits} \tag{3.16}$$

$$P^{icache}_{Switching\_est} = \left[\frac{E^{itram}_{Switching\_est_{read}} \times (N_{itram,write} + N_{itram,read}) + E^{icram}_{Switching\_est_{read}} \times (N_{icram,write} + N_{icram,read})}{simtime} \right] \tag{3.17}$$

Where:

$Width_{tbits}$: is the width of instruction tag ram

$Width_{icbits}$: is the width of instruction cache ram

$N_{itram,reads}$: is number tag ram reads

$N_{itram,write}$: is number tag ram write

$N_{icram,reads}$: is number cache ram reads

$N_{icram,write}$: is number cache ram write

$simtime$: is design simulation time

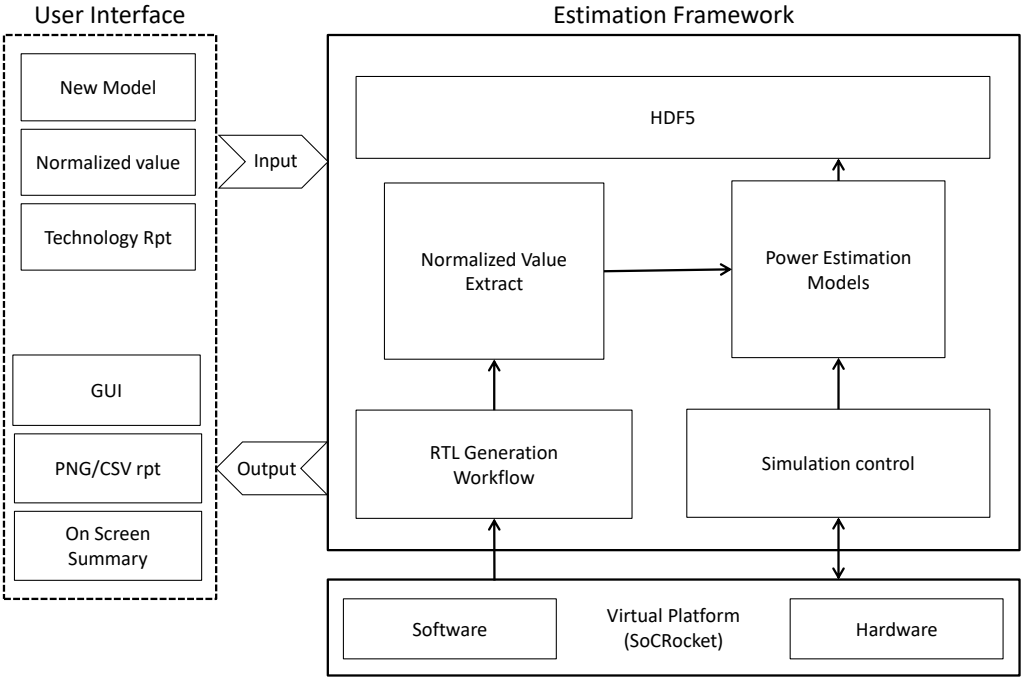


Figure 3.3: Power Estimation Framework and User Interface

3.2 Power Estimation Framework

The presented power estimation models are based on the system-level event-triggered power-monitoring concept. Figure 3.3 shows the power estimation framework around SoCRocket, the power estimation framework is separated from the SystemC platform and it built in Python. Power estimation can be done inside SoCRocket by passing normalized power to values into the simulation via JSON configuration file, however, this approach lacks long-term flexibility, as modifications to the power models are a labor-intensive job. The whole platform would have to be recompiled whenever a single change is made to a model. Therefore, a separate estimation framework approach was chosen. The Framework contains python scripts, which can be easily modified without rebuilding the platform.

3.2.1 Simulation Control

The SoCRocket USI scripting interface enables read and write accesses to model parameters during simulation. Using USI scripting interface, a python script ”simCtrl” is

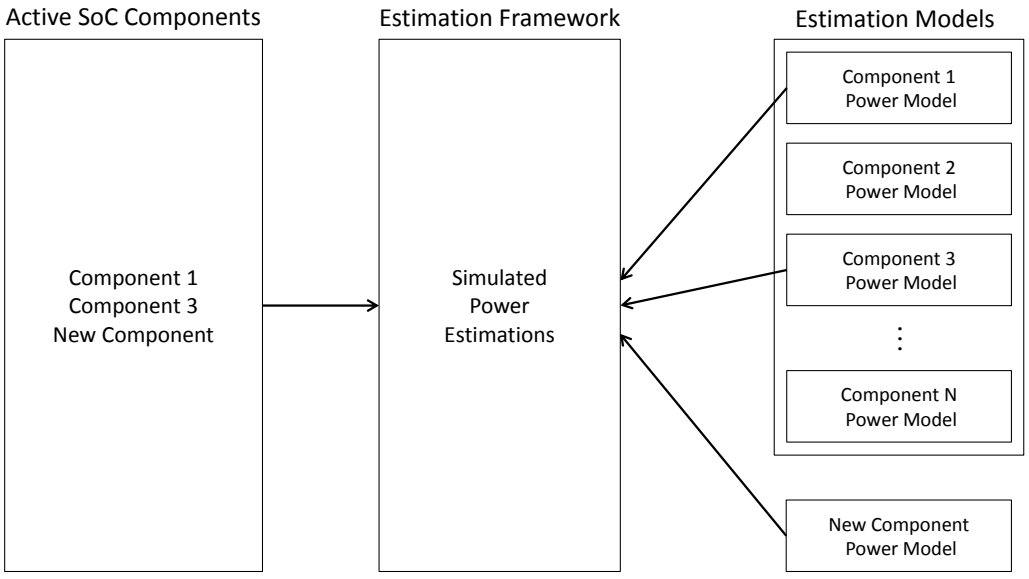


Figure 3.4: Integration of Power Models

applied to steer the SystemC simulation, which means running the simulation in start-pause-start order. The ”simCtrl” retrieves simulation data and events (e.g. the number of accesses), which are read at the end of a user-specified simulation interval Δt. This is used to calculate the instantaneous power draw. This process is repeated until the execution is completed. When Δt is not defined than the whole simulation is run without interrupts. Also, users can terminate the simulation with a specified time or by invoking a corresponding SIGINT-signal. This allows a user to get control on lengthy simulations that eases debugging process.

3.2.2 Dynamic Integration of Power Models

As shown in the Figure 3.4, the estimation Framework allows dynamic linking of power models for active components which enables a smooth integration/loading of the models into Framework simulation setup. The metadata of system components is collected from the virtual platform and only the active components are loaded into estimation Framework during simulation, which gives freedom to the system-level designer to enable or disable a component during the architectural exploration in order to monitor its effect on system power draw. When hardware software exploration (this is topic will be covered in Chapter 5) is carried out and a new hardware accelerator is integrated into the existing

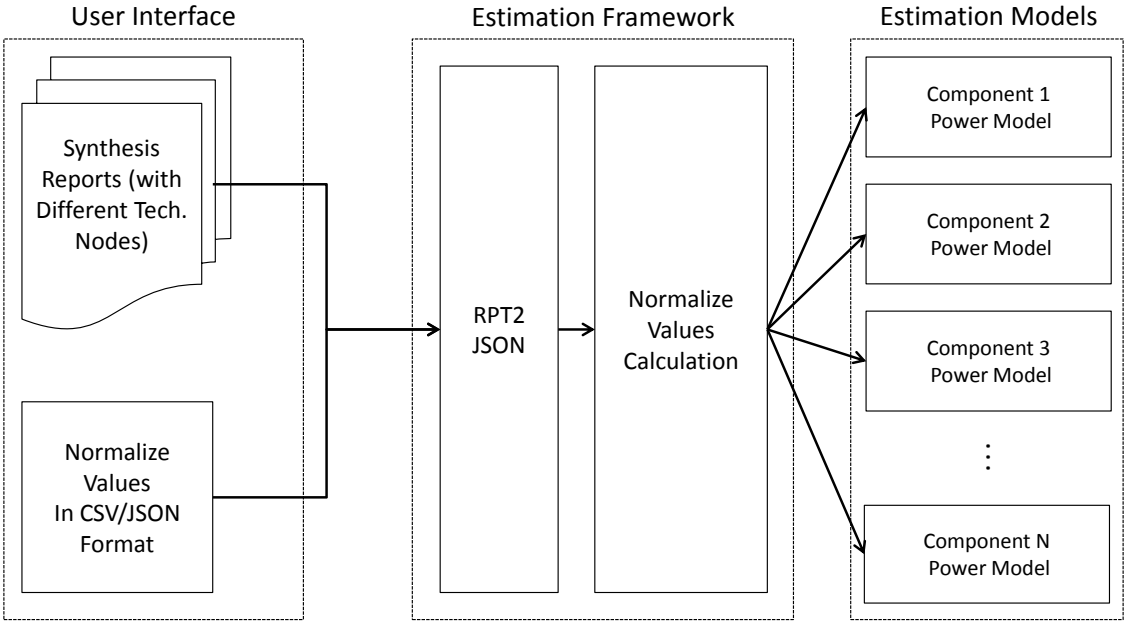


Figure 3.5: Update of Normalized Power Values

SoC architecture, the Framework allows a seamless creation of a new power estimation model and its ingratiation into the existing ones.

3.2.2.1 Normalized Power Values Generation for New Technology Nodes

To reflect the changes in technology nodes at system level architectural exploration, designers need to start over the whole estimation modeling process i.e. perform low-level analysis and create new estimation models. The same applies to power numbers extracted from hardware prototyping (e.g. FPGA), to see their impact on the system level, an update of the models is required. The presented power estimation framework incorporates technology nodes changes and allows the designer to update or expanded quickly the pre-existing power values without rebuilt of the virtual platform.

Figure 3.5 shows, a designer can specify a new RTL simulation synthesis report with changed target technology node. The "RPT2JSON" tool converts this report into a JSON file, which is used for the calculation of new normalized power values. This simplifies appending or changing the existing normalized power values. The extracted power values can be supplied as a CSV spreadsheet (which will be converted to JSON) or directly as a

JSON (as shown in Listing 3.1), the tool will update the estimation models using the new values.

```
{
  "power": {
    "leon3": {
      "sta_power_norm": 5e+4,
      "int_power_norm": 5e−10,
      "dyn_instr_energy_norm": 4e−9
    }
  }
}
```

Listing 3.1: Example JSON file for Power Values

3.2.3 System-Level Instantaneous Power Representation

Since average power represents the consumption of energy over time, it is not a sufficient unit for system-level analysis. For example, hotspots cannot be identified when only looking at average power consumption. To this end, an instantaneous system-level power analysis tool is presented in this work. The granularity of instantaneous power reporting can be set Δt in Equation 3.11. Smaller Δt gives more accuracy but simulation speed will suffer because of large amounts of simulation data (power reporting of every component and logging etc.) is generated stored into a database.

HDF5 database is selected for storing the simulation data fostering easy and speedy power analysis. The database itself was designed for complex and large datasets. It can handle a large amount of data and can be easily integrated into the Python environment. To filter out unwanted logging output of virtual platform, a white- and blacklist filtering approach [43] is used.

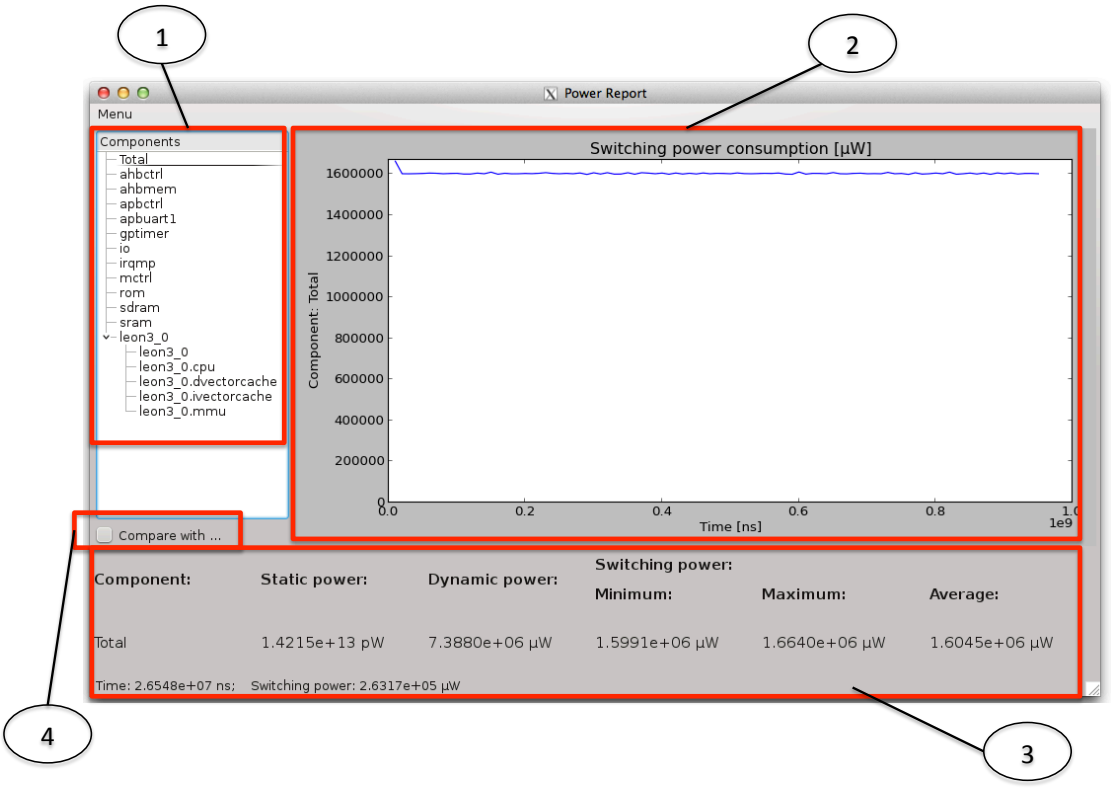


Figure 3.6: Power Reporting GUI

3.2.4 Power Reporting

The efficiency of a power estimation framework relies on usability, a Qt-based GUI was designed in this work to allow easy user interfacing to aid efficient DSE. As shown in Figure 3.6, the GUI has four main features:

1. Easy navigation through the components lists. As shown in the figure, the user can select any component or subcomponents with a simple drop-menu on the left side of the screen.

2. The instantaneous power view window can be seen on the right side of the screen, it shows an extrapolated plot of the power against simulation time for the selected component. Accuracy depends on the chosen Δt.

3. A power summary is shown in the lower part of the screen, it reports the component name, static power, dynamic power, switching power (peaks and average) and total

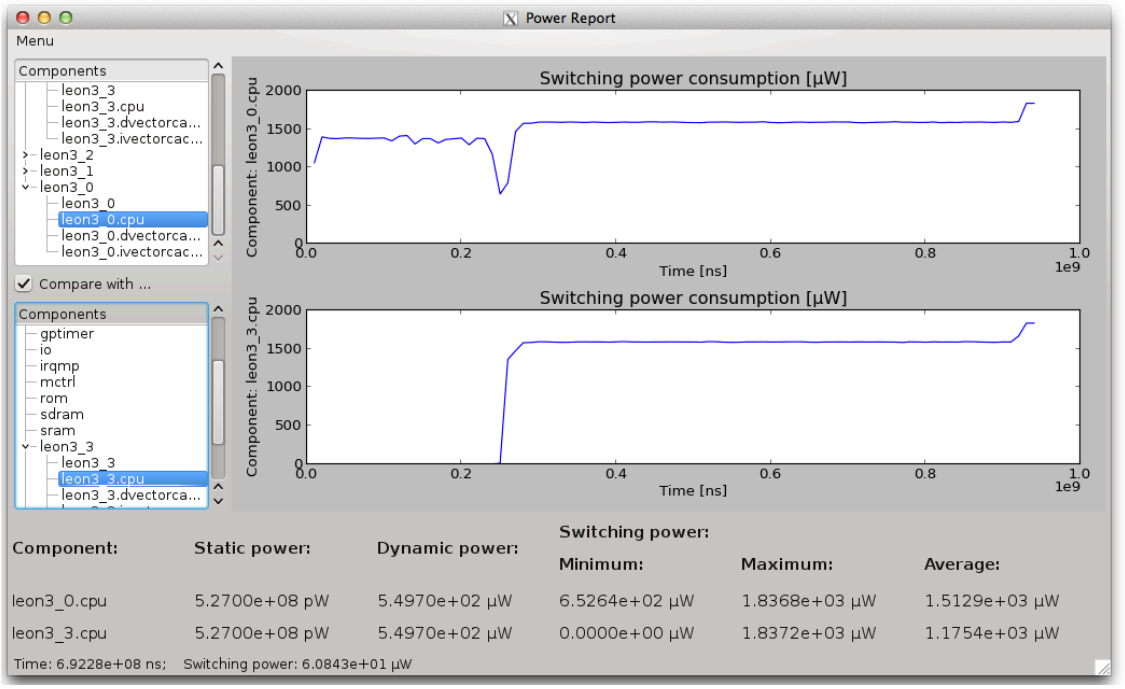


Figure 3.7: Power Reporting GUI: Components Power Draw Comparison

simulation time. This allows a designer to detect the hotspot early in the design phase.

4. Comparison of power draw between two components can be viewed at once via a split screen option. In Figure 3.7 , four Leon3 based multicore system components are shown. Leon3_0 is the master core which is running an operating system while the other three are bare-metal cores. The power consumption comparison can be seen among the cores and the system components of this MPSoC. For example, as it is shown in the figure ($0ns - 0.24ns$), the Leon3_0 core is consuming more switching power while it is booting the operating system compared to the Leon3_3 core.

Listing 3.2 shows an example, how to configure different parameters of the power reporting, like enabling/ disabling writing to csv, the GUI or the textual summary. If power reporting is used with the GUI, the GUI allows saving the results to HD5 file. This file can be reopened by the GUI without running the simulation beforehand. That way, results may be stored and revisited at a later analysis. Since the format used by the GUI is the same as the one used by the simulation itself, results of earlier simulations may also be

viewed with the GUI. It is also possible to export the results as a $.csv - file$ and graphs (as PNG).

```
{
  "powerreporting": {
    "enable_summary": false,
    " enable_report_over_time ": true,
    "enable_csv_output": true,
    " enable_gui_report ": false,
    "enable_png_output": true,
    " power_reporting_interval_ns ": 1000,
    "power_reporting_endtime_ns": 0,
    " output_folder ": "./ result_dir ",
    " hdf_output_filename ": "my_own_hdf_file.hdf",
    " csv_output_filename ": "my_own_file.csv",
    "model_paths": {
      "accel_dev":"./models/accel_dev"
    }
  }
}
```

Listing 3.2: Example Power Reporting Configuration

3.3 Summary

This chapter described the static power and the dynamic power which are two major components of total power dissipation of an electronic design. The system-level power estimation based on the event-triggered power-monitoring concept was explained. A workflow for RTL code generation from a SystemC virtual platform configuration was shown. RTL code later was used to calculate the normalized values of presented power estimation models. Finally, a detailed overview of the presented power estimation framework was given in this chapter.

Chapter 4

Design Space Exploration Hardware

In the design of MPSoC choosing the right architectural parameters is a very important task, adjusting these parameters configurations produce very different results in the performance and the power consumption of the system. Tuning of design parameters for the required design constraints is a multi-objective optimization problem which involves either maximizing or minimizing the cost function of the complete design space. Typically, the solution of such an optimization problem is not a single point solution rather a set of optimal solutions which lie on the Pareto curve [115]. Traditionally, the design space exploration of finding the Pareto curve for the given MPSoC design parameters is depended on the designer experience and requires a large number of simulations to be performed in the process. Due to the rapidly growing architectural complexity of SoC, it is not feasible to simulate all possible input parameter permutations. This chapter presents the design, development and implementation of a fully automated DSE framework for the MPSoC architectures using a SoCRocket virtual platform.

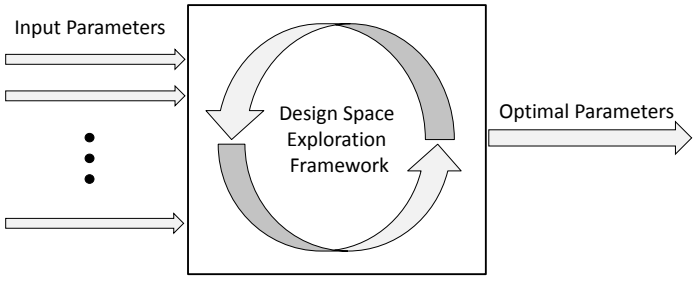


Figure 4.1: Block Diagram for Design Space Exploration Framework

Figure 4.1 shows the block diagram illustrating the functionality of the presented DSE framework. Using architectural parameters as inputs, fast and accurate simulation models are applied to identify design parameters with the optimal characteristics for various design goals. Powerful optimization strategies inspired by simulated annealing and the evolutionary genetic algorithm are employed in the framework to reduce the design exploration time. Following sections describe presented optimization techniques along with the DSE framework.

4.1 Architecture Design Space Exploration Methodology

The design space is determined by the number of parameters and their set-size under the exploration subjected to the system requirements and constraints. In Equation 4.1, $N = |D|$ shows the total number of design points (size of design space) to be explored for M design parameters as given in Equation 4.2.

$$D = \{P_1, ..., P_N\} \tag{4.1}$$

$$P_i = \{p_1^i, p_2^i ..., p_M^i\} \tag{4.2}$$

where p_j^i is the value of j^{th} parameter (i.e. 1 to M) for the i^{th} point in the design space D.

Figure 4.2 shows presented DSE methodology. The presented methodology takes the set of architectural parameters for exploration defined by the designer. The DSE process starts with an initial parameter configuration. If not provided by the designer, the initial configuration will be selected randomly from the design space.

Then the design space exploration begins and the simulation results of the objective functions (system-level metrics) are stored in the database. The DSE optimization techniques may result in repeated simulations of a parameter configuration. To avoid unnecessary simulation overhead, the database is queried for the result of the past simulation for the selected configuration. If the result is missing, then the configuration is simulated using the virtual platform simulator and the output is stored in the database.

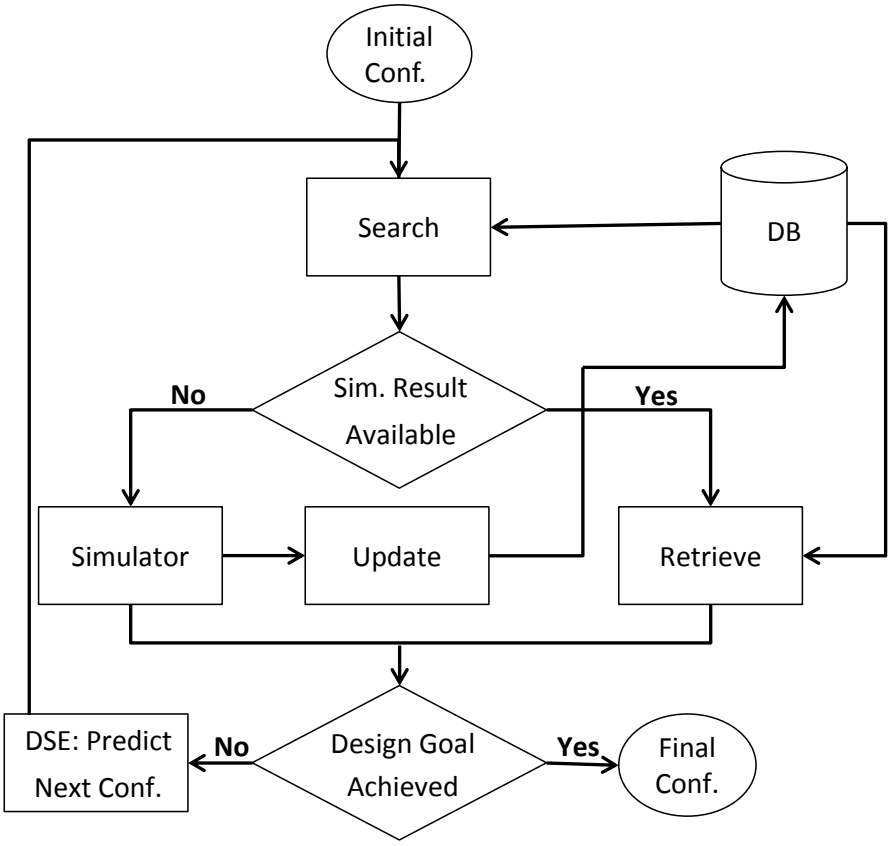


Figure 4.2: DSE Methodology

When the desired cost (design goal) is achieved, the final parameter configuration is reported and the process is terminated. If the desired design goal is not achieved, the DSE framework predicts a new configuration and the process is iterated until the promising results are obtained. In the following, components of the presented methodology are discussed in detail.

4.1.1 Architectural Parameters under Exploration

The parameters such as the number of processor cores and cache configuration are essential architecture characteristics defining the performance and power consumption in modern MPSoC. An example of the design space is shown in Table 4.1. The parameter names, their descriptions, and the possible values they can take on are defined. It is important to mention that the presented framework in this dissertation is not limited to these

Parameter	**Set**	$\|Set\|$
Number of Processors	p_1	4
Instruction Cache Banks	p_2	4
Instruction Cache Bank Size	p_3	5
Data Cache Banks	p_4	4
Data Cache Bank Size	p_5	4

Table 4.1: An Example of Optimization Parameters

parameters, it is designed to be flexible and can be extended to the other architecture parameters e.g. memory configuration etc.

In the current example, there are 1280 possible configurations (total combinations of parameters in Table 4.1), which is also considered as the size of the problem space. Conventionally, the designer needs to explore the entire problem space to find the best design point for a given design goal. Chapter 6 will elaborate the mapping of the target design of space to the design decision metric in terms of accuracy and performance.

4.1.2 Architecture Cost Evaluation Model (Decision Metric)

The objective of the SoC design depends on the target application. On the one hand, critical applications demand high performance, while on the other hand, system power consumption is an issue when the battery life is concerned. Keeping in mind these two important issues of modern SoC design, the Energy Delay Product (EDP) is chosen as a possible metric to search for an optimal design. This means defining the best set of configuration parameters in terms of execution time and power dissipation of the system. The range of optimal results can be also specified (e.g. $< minEDP, maxEDP >$) as a decision criterion. The presented framework also allows the designer to change or use other performance metrics in the cost functions (e.g. area etc).

4.1.2.1 Termination Condition

In addition to the cost evaluation model described in the previous section, the DSE process can be terminated after a given number of maximum simulations is reached. In this case (depending on the number of simulations), the presented framework may not find

the ”sufficiently good” result, which is close to the optimum values of the parameters. However, in practice this stop condition is very useful to give the designer a quick insight of very large design space with reasonable accuracy.

4.2 DSE Optimization Strategies

Exhaustive search (ES) creates all possible combinations of parameters from the design space and runs the corresponding simulations. In contrast to other DSE algorithms, it is not designed to stop once a sufficiently good configuration is found. It will instead run all the simulations and at the end return the best possible configuration. The total number of simulations required can be calculated with Equation 4.3. Where M is the total number of parameters under explorations and $|p_i|$ is total number of p_i set elements.

$$ES_{total\_simulations} = \prod_{i=1}^{M} |p_i| \tag{4.3}$$

Since every possible combination is evaluated, the result is the global minimum. Depending on application and number of parameters, ES can take a large amount of time (often days). Which leads to an extensive delay in system level analysis. In order to speed up the DSE process, the presented DSE Framework integrates several fast exploration strategies to find the optimal parameter configuration. Results discussed in Chapter 6 shows, the strategies presented in the following sections achieve a desirable configuration with the smaller number of simulations compared to ES.

4.2.1 Random Optimization

The random search optimization algorithm (RND) takes a start configuration and saves it as the current best configuration (P_{best}) as described in the Algo. 1. The optimization runs in a loop until the termination condition is met. In each iteration of the loop, a new configuration P_{new} with M parameters is generated by randomly replacing every parameter in the configuration P_{new} either by itself (0) or one of its direct neighbors (-1/+1). The first iteration’s configuration stored as the current best result. After every

iteration, it is checked whether the new result is better than the current best. If it is, the current best is replaced by the new configuration. Also, it is checked whether the new result is sufficiently good means it fulfills the design requirement then the algorithm terminates. Otherwise, the search goes on. If a reasonably large number of possible configurations are run without finding a sufficiently good solution, the loop is stopped and the current best result is returned. In the extreme case, RND behaves like the ES, since it can end up exploring all the design points while searching for the best solution.

Algorithm 1 Random Search

$P_{best} = P_{init}$
while `(true)` **do**
  $P_{new} \leftarrow P_{best} + random\_direction(\{-1, 0, +1\}, M)$
  $P_{best} = compare\_replace(P_{new}, P_{best})$
  if $Termination\_Condition$ **then**
   `break`
  end if
end while

4.2.2 Single Parameter Optimization

The Single Parameter Optimization (SPO) is shown in Algo 2, in this algorithm, it is assumed that all parameters are independent of each other. Then every parameter (p_j) is optimized on its own, not changing any other parameter in the given start configuration (P_{start}). The inner loop runs k_j times, which is the number of values p_j can take in a given design.

Algorithm 2 Single Parameter Optimization

$P_{start} = \{p_1^1, p_1^1 ..., p_M^1\}$
$P_{best} = P_{start}$
for `j:=1 TO M STEP 1` **do**
  for k_j `times` **do**
    $P_{new} \leftarrow P_{best} + direction\_of\_param\_j(+1)$
    $P_{best} = compare\_replace(P_{new}, P_{best})$
    if $Termination\_Condition$ **then**
   `break`
    end if
  end for
end for

Finally, the best values chosen for each parameter are merged into a single configuration and this is returned as the overall all best result. If during the optimization of a single parameter a sufficiently good result is found, then the algorithm returns and the rest of the search is skipped. In every iteration of the algorithm, if the newly generated result is better than current best (for a given parameter) then the reference result (P_{best}) is updated. If the last tested configuration matches the termination criteria, the algorithm returns the corresponding configuration as the best possible configuration. If the termination criterion is not reached, the configuration closest to the termination cut-off is selected. In reality for most multi-processor applications, the design parameters such as the number of cores and cache banks and size are not fully independent. In fact, the 2x and 4x multi-processors do benefit from larger cache sizes. The total number of simulations required (when the termination criteria are not met) can be calculated with Equation 4.4.

$$SPO_{total\_simulations} = \sum_{i=1}^{M} |p_i| - M \tag{4.4}$$

Algorithm 3 Multi Parameter Optimization

$P_{best} = P_{init}$
while `(true)` **do**
  for d_1 `:=-1 TO +1 STEP 1` **do**
    .
    .
    .
    for d_M `:=-1 TO +1 STEP 1` **do**
      $P_{new} \leftarrow P_{best} + \{d_1, ..., d_m\}$
      $P_{best} = compare\_replace(P_{new}, P_{best})$
    end for
    .
    .
    .
  end for
  if $Termination\_Condition$ **then**
   `break`
  end if
end while

4.2.3 Multi Parameter Optimization

The Multi-Parameter Optimization (MPO) is basically an exhaustive-search at a local level. During each iteration of the Algo. 3, all parameter configurations are searched in three possible directions during the DSE process: ($d_i = -1$) backward, ($d_i = 0$) no-change and ($d_i = +1$) forward. Therefore, each parameter has three possible values in the reduced design space, giving 3^M variations per ES (as shown in Equation 4.5), where M is the number of parameters.

$$MPO_{single\_iter\_simulations} = 3^M \tag{4.5}$$

After each ES, if the resulting parameter configuration matches a termination condition then it is returned as the desired design configuration. If it matches the former best result, it means optima are found and the DSE process is terminated. If these two conditions are not met, a new ES is started, with the current best configuration as the start configuration. The number of simulations in a single iteration of the MPO (Equation 4.5) looks quite large, but in practice, the final number of simulations is much less due to repeated configurations. Since a database of simulation results is kept, the repeating configurations do not need to be re-evaluated.

Algorithm 4 Fast Multi Parameter Optimization

$P_{best} = P_{init}$
while `(true)` **do**
  for `j:=1 TO M STEP 1` **do**
    $P_{new+1} \leftarrow P_{best} + direction\_of\_param\_j(+1)$
    $P_{new-1} \leftarrow P_{best} + direction\_of\_param\_j(-1)$
    $P_{best} = compare\_replace(P_{best}, P_{new-1}, P_{new+1})$
  end for
  if $Termination\_Condition$ **then**
   `break`
  end if
end while

4.2.4 Fast Multi-Parameter Optimization

Fast Multiple Parameter Optimization (FMPO) is an optimization to the MPO algorithm. Starting with an initial (given) configuration, it enters an endless loop as shown in Algo. 4. In each iteration of the loop, the parameters of the simulation are changed as follows:

- The first parameter in the current best configurations is either kept or replaced by one of its direct neighboring parameters (-1/+1).
- From these three configurations, the best is chosen as the current best configuration.
- The same process repeated for the second parameter until the last parameter in the current best configurations is updated.

After each loop iteration, it is checked whether:

(a) a sufficiently good solution was achieved. If so, it is returned as the desired configuration.

(b) no more progress is made. If so, either local or global optimum of parameters is found.

The number of simulations in a single iteration of the FMPO can be calculated with Equation 4.6. This algorithmic simplification shows a significant reduction in the total number of simulations (from 3^M to $3 * M$) in the comparison of MPO. It seems like a serious limitation on the optimization process, however, this simplification performs quite reasonably in general on the multi-processor benchmarks. Further discussion on this issue is presented in Chapter 6.

$$FMPO_{single\_iter\_simulations} = 3 * M \tag{4.6}$$

4.2.5 Genetic Algorithm based DSE

The genetic algorithm is well known in the computing world, first invented by John Holland in 1960, a heuristic method of evolutionary computing, the idea of evolutionary computing is inspired by natural evaluation. Holland's work provides a theoretical basis for

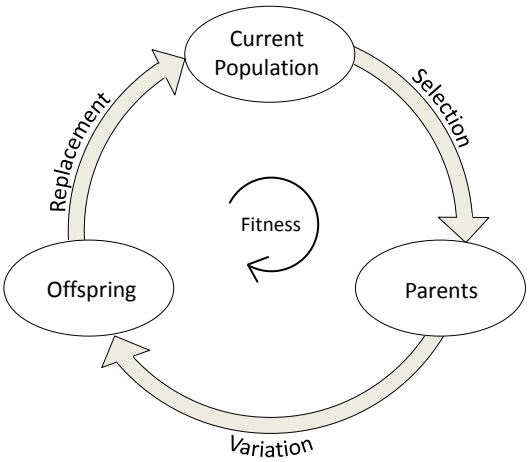


Figure 4.3: Basic cycle of Evolutionary Computing

applying biological evolution to solve real-world optimization problems [116][117] that includes numerical optimization, stochastic programming, ecological modeling, macromolecular structure prediction and many more in the various fields of modern science. In the biological evolution, the natural selection plays a key role in species survival in competitive environments, survival of the fittest means the genes of the stronger individuals, who can adapt according to their surrounding environment, are passed to next generation. This survival of the fittest approach is mimicked under genetic algorithm, it allows a massive parallel search by simultaneously changing the numerous design points in parallel directions rather than working on one point in a single direction. This optimization technique permits to find global minima for highly non-linear bigger design space.

4.2.5.1 Component of Genetic Algorithm

As mentioned in the previous section, survival of the fittest nature makes the evolutionary computing a good candidate to solve the optimization problems of type $generate-and-test$. Figure 4.3 shows the basic cycle of evolutionary computing. The components of evolutionary computing define the transition from one state evaluation to another one, as shown in the figure, they define the quality of the evolutionary algorithms, in the following sections the important components of the genetic algorithm are defined based on [118].

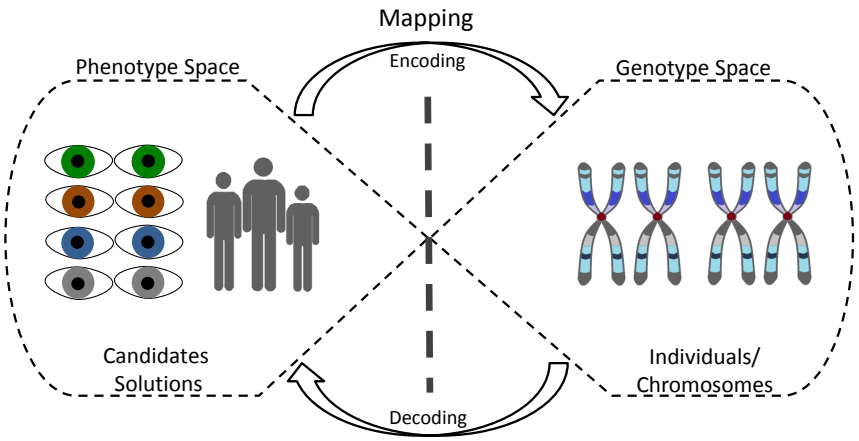


Figure 4.4: Representation/(Inverse Representation)

4.2.5.2 Representation

Every organism has a group of distinguishable, observable and obvious features (like eyes color, hair color and height etc), in biological terms these characteristics are called phenotypic trait. These visible characteristics are results of different DNA genes sequences (known as genotype) and the influence of the surrounding environment. In simple words, phenotypes are represented by genotypes. When this analogy is applied to an evolutionary computing algorithm then first design step is called ”representation”. Figure 4.4 shows the phenotype space, a set of individuals (also called candidate solutions), is represented by encoding of genotype space, a set of individuals (also called chromosomes). The decoding of genotype space to phenotype space is called ”inverse representation”. Each individual of phenotype space is a unique combination of genotypes. In the evolutionary computing algorithm, depending upon search problem there are several possible types of representation e.g binary, integer, real value of floating point, permutation and tree.

The presented DSE framework envisages the system to find the optimal design values of the architecture parameters in MPSoC design. The configurations in this design space have integer parameters values. The straightforward approach is to represent these parameters by a finite set of integers without further encoding. This approach makes the phenotype space same as the genotype space. Each configuration of parameters in Equation 4.1 is called chromosomes and value of each parameter in Table 4.1 is called gene.

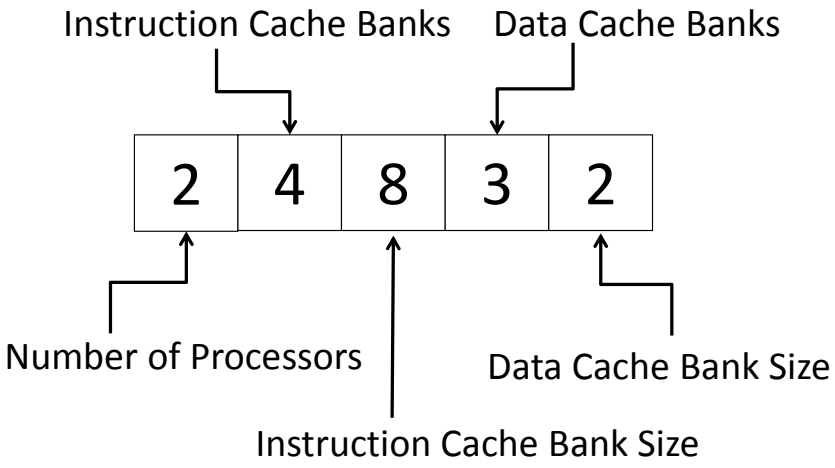


Figure 4.5: An Example of Chromosome (individual) with 5 Genes (Parameters)

Here a chromosome (as shown in Figure 4.5) is an ”individual” which represents the possible solution of the design space.

4.2.5.3 Fitness Function

In genetic algorithm fitness function also called evaluation function is a criterion to determine the fitness of individuals, whether these individuals are suited to the surrounding environment or not. It provides the basis for better next generation selection. The proposed fitness function is already is described in the preceding Section 4.1.2. The main goal of each iteration (generation) of the genetic algorithm is to get closer and closer to this fitness function. The designer can choose among the group of individuals with most fitness for the optimal design

4.2.5.4 Population

Different individuals make a set of possible solutions knows as population. The population forms the evolution unit, its size stays constant throughout genetic algorithm search however its composition is changing with every generation. Depending on the requirement of algorithm application, there can be a duplication of individuals (allowing redundant individuals). If the duplication policy is not adopted, then every individual must be

unique in the current population. The size of a population is proportional to the search space size. There are mainly two types of models for population management:

- **Generational**: Each iteration has a new group of individuals means the whole population is replaced by the new population.
- **Steady-State**: Instead of changing the whole the population only some individuals are replaced with the new individuals.

4.2.5.5 Parent selection

It is a selection of two individuals as parents, they reproduce new individuals called children. The group of children is also called offspring. The selection of an individual to be a parent is determined by its quality of fitness, stronger individuals have more chance to become the parent as compare to the weaker individual. In order to keep diversity in the algorithm (to avoid local minima), the weaker individuals must not be totally discarded from the population. There are many methods of selection, some popular parent selection approaches [119] are:

- **Elitist selection**: This selection is to preserve the best individuals, first or few best fit individuals are selected.
- F**itness proportional selection**: The selection probability is obtained by diving the absolute fitness value by the absolute fitness value of the rest of the population (e.g. roulette-wheel selection)
- **Ranking selection**: The individuals assigned by numerical ranks according to their fitness. The population is sorted based on the rank value, which means an individual is selected on the difference of rank value rather than absolute fitness difference.
- **Tournament selection**: The selection is used when the whole population is too large and entire knowledge of the population is not available, subgroups of individuals are picked from the population and the best individuals are selected from each subgroup.

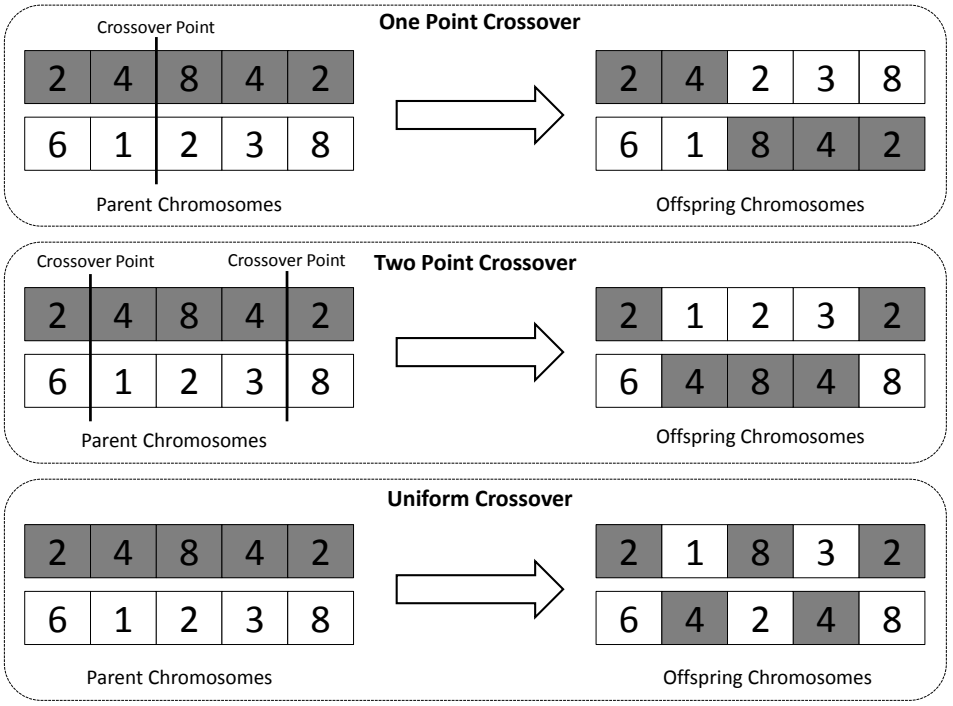
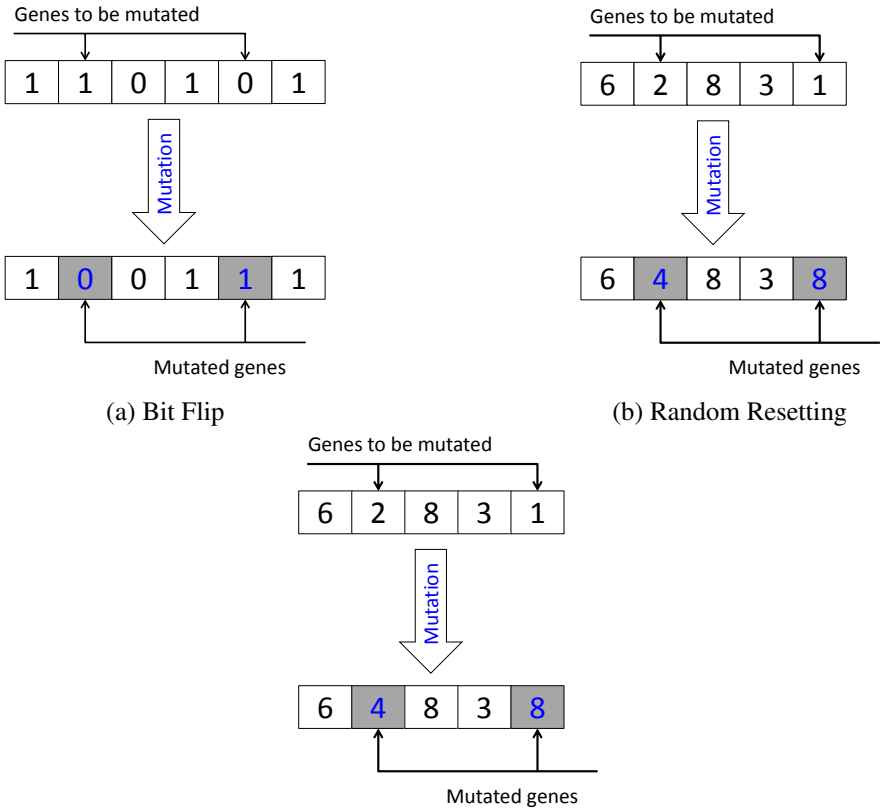


Figure 4.6: Crossover Types, This figure is reproduced from [120]

4.2.5.6 Variation Operators

After the selection process is done and best fit individuals are chosen for a population, some individuals and randomly picked and alerted to increase the chances of a new generation with better fitness. There are two main variation operators:

- **Crossover:** This operator is also called recombination. It is about exchanging the information(genes) of two parent individuals to produce two new individuals (children). Figure 4.6 shows the three common types of crossovers: **one point crossover**, a point is randomly set, beyond this point all genes are exchanged between two individuals; **two points crossover**, all genes of two individuals exchange between start point and end point (it is also called multi point crossover when there are more than two segments needed to exchange among multi points); and **uniform crossover**, instead of segments any gene can be exchange(or stay its own position) between to individuals with 0.5 probability.



(a) Bit Flip

(b) Random Resetting

(c) Creep Mutation

Figure 4.7: Examples of Mutations

- **Mutation:** This operator provides diversity in the algorithm and it prevents the local optimum problem. From one chromosome, mutation produces a new chromosome by distorting (interchanging) the value of its genes with a certain probability (p_m, known as mutation probability). There are many types of mutations, depending on the type of the chosen representation, different mutation is applied. For example, in evolutionary computation the most commonly used is **bit flip**, it is applied to binary representation as shown in Figure 4.7a, its simple flipping the value $(0/1)$ of the chosen position for mutation.

  For the integer representation there are mainly two types of mutations: **random resetting** (see 4.7b); is similar to bit flipping instead of flipping the value of integer (gene) is replaced by another random integer, which is selected from the allowed values on that position, this is used for cardinal attributes (non-ordinal attributes); and **creep mutation** (see 4.7c) is adding a small value (either negative or positive)

to particular gene (symmetrically distributed around 0), this operator is applied to ordinal attributes.

The mutation probability for each allele (position) of the chromosome depends on nature of the problem, typically it is in the interval $[1/population\_size, 1/chromosome\_length]$.

4.2.5.7 Survivor Selection or Replacement

The survival selection describes who will carry-on to the next generation. It is also called replacement because the individual failing to survive will be replaced by a new individual. An ”age-based” replacement means that all the offspring will replace the actual (parental) population as a new generation (similar to First In First Out), it is an example of the generational population management model. The replacement can be ”fitness-based”, the offspring and parental population are ranked according to their fitness, best of them will become a new population, this is a steady-state population management model.

4.2.5.8 Stop Condition

Like every algorithm, stop condition is very important and one of the structural parameter of genetic algorithm [121], it deeply affects the convergence of the algorithm. There is no general rule of defining an adequate stopping condition for the genetic algorithm, Nerveless, the Section 4.1.2 can be used to define the stop condition for the genetic algorithm as:

- **Limited Generations**: In this method, the total number of generations is fixed by the designer. When this limit is reached, the algorithm stops.
- **Limited Simulations**: There can be a duplication of individuals over the generations, in this method the total number of unique individuals (simulations) is fixed. When this number is achieved then the algorithm terminates.
- **Optimal Fitness Threshold**: There is no limitation on the number of generations. However, the designer fixes an interval for the best parameters quality (e.g. best EDP) which is related to the fitness function. When the fitness of an individual of the current generation lies in this interval, the algorithm stops.

After terminating the algorithm, the presented framework chooses the individual with best parameter configuration among the list of the individuals with optimal fitness, in every generation, individuals with the best fitness are stored in a list called $best\_group$, and when the algorithm stops, the proposed framework chooses the individual with best parameter configuration among the list as final output.

Algorithm 5 Fast Genetic Search based on [118]

```
0. Initialize Population
0. Evaluate
while (true) do
    1. Parent Selection
    2. Crossover
    3. Mutation
    4. Evaluation
    if Termination_Condition then
        break
    end if
    5. Replacement
end while
```

4.2.5.9 Fast Genetic Search

The genetic optimization algorithm (shown in Algo.5) works as follows: In the first step an initial population is chosen from the whole design space and random configurations are created for this population size (given by the user). Then, the best configuration of these is determined. If that matches the 'sufficiently good' criteria, it is returned. If that is not the case, the optimization process starts. From the whole created population, $2 * < tournament\_size >$ candidates are randomly selected; from each tournament, the best becomes one parent. From these parents, two children are created. The first child inherits its configurations up to a crossover-point from the first parent and from the crossover-point on from the second parent. The second child gets the remaining configurations. After that, the children are subject to a random mutation, possibly changing parameters in their design by replacing them with random (but valid) values from the design space. This is done with probability p_m (Equation 4.7), so a new generation the same size as the old one is created.

$$p_m =< population\_size > /2 \tag{4.7}$$

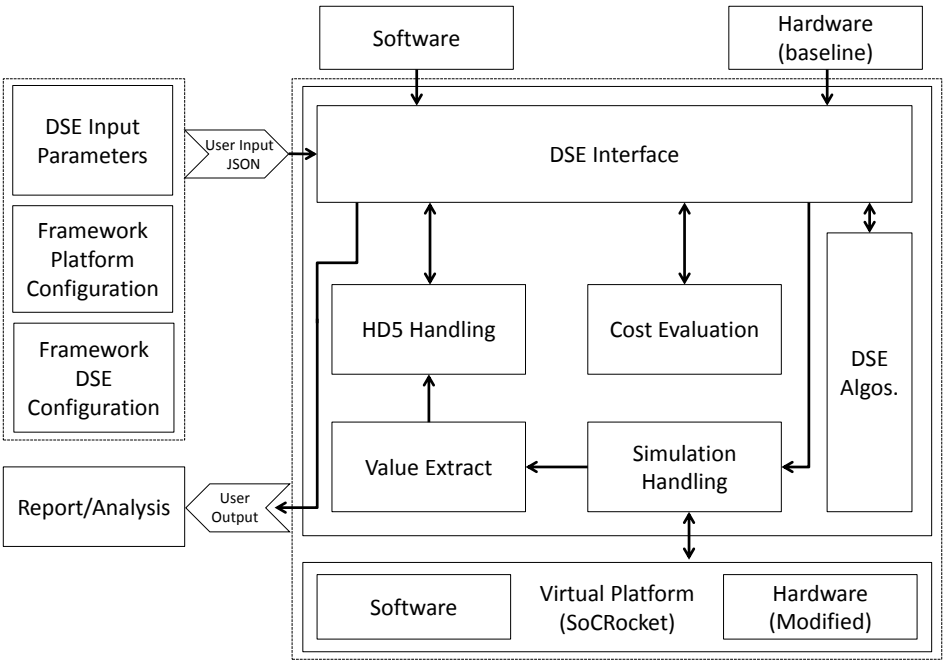


Figure 4.8: Automated Design Space Exploration Framework

After the new generation is created, the current best configuration is determined and several termination criteria have to be checked:

(a) If a sufficiently good configuration was found, that one is returned

(b) If the maximum number of generations was reached, the current best result is returned

(c) If the maximum number of unique simulations run is reached, the current best result is returned.

If none of the three options above are true, a new generation is created again and the whole process is repeated.

4.3 Automated Design Space Exploration Framework

Reducing the effort and time needed in exploring the large design space of MPSoC design are the main goals of the proposed DSE framework. It hides the complexity of the

process and gives the designer more flexibility to choose the best parameters for a given design specification. Figure 4.8 shows the tool flow and user interface. The framework runs design space exploration with the SoCRocket environment. It takes a number of arguments to configure the simulation environment. The usage of the framework tools is explained in the following sections.

```
{
  "conf": {
    "mmu_cache": {
      "ic": {
        " setsize ":  [1,2,4,8,16],
        " sets ":  [1,2,3,4]
      },
      "dc": {
        " setsize ":  [1,2,4,8,16],
        " sets ":  [1,2,3,4]
      }
    },
    "system": {
      "ncpu":  [1,2,3,4,6]
    }
  }
}
```

Listing 4.1: JSON configuration file for parameters

4.3.1 User Input (JSON)

Input DSE Parameters: SoCRocket virtual platforms configuration can be manipulated without changing the source code which allows fast and easy modifications to the platform configuration without requiring lengthy compilation cycles. These platform parameter configurations are encoded in the JSON file format or given as command-line simulator argument, assigning values to different parameters defined in the top-level configuration. Parameters which are under exploration (see Table 4.1) are given to DSE

automation module. An example of the DSE input parameters JSON file is shown in Listing 4.1.

Parameters belonging to specific modules are added to this array (Line 2), which itself is a parameter array with the string name $mmu\_cache(icanddc)$ and $system$ respectively. Each element (e.g. $conf->system->ncpus$) represents one possible parameter of the design space. The corresponding list shows which values are valid for a parameter (e.g. ncpu: 1, 2, 4, 6).

```
1  {
2    "platform": "path_to/leon3mp.platform",
3    "elf": "path_to/software.sparc",
4    "template": "path_to/hardware_template.json",
5    "scripts": [
6      "path_to/power_reporting.py",
7      "path_to/report_config.json"
8    ],
9   "intrinsics": ""
10 }
```

Listing 4.2: Framework Platform Configuration JSON file

Framework Platform Configuration: Listing 4.2 shows an example of a platform configuration which contains:

- The location of the executable simulator platform
- The path to the application executable (elf) to run on the simulated SoC
- The path to the template to use for the hardware configuration ($template.json$)
- Paths to the additional scripts (e.g. $power\_reporting.py$) to access the power estimation framework
- The configuration files ($report\_config.json$) to filter out unnecessary warnings and reporting messages during simulations.

Platform intrinsics are given by adding the parameter "$intrinsics$", simulator emulates libc IO functions (e.g. printf) to simplify/accelerate the simulation of a complete system.

Framework DSE Configuration: The name of a specific algorithm and its specifications such as the maximum number of simulations (if it is not given then it is the total number simulation resulting from all possible configurations) are given in Listing 4.3. The genetic algorithm components can also be specified in the JSON file. Otherwise, the default component values will be selected for the algorithm. These are: population size is 20 (must be a multiple of 2), tournament size is 7, mutation threshold is 0.5 and max number of generations is 50. When the field "$absolute\_min\_en$" is enabled then the best optimal design point (e.g minimum EDP) is searched (worst case is ES), alternatively the user can provide the desired range of solutions ($< minEDP, maxEDP >$) as a design criterion.

```
{
  " algo_specification ": {
    "algo_name": "",
    "max_number_simulations": "",
    " population_size ": "",
    "tournament_size": "",
    " mutation_threshold ": "",
    "max_number_generations: ""
  },
  " cost_function ": {
      "absolute_min_en": true ,
    "max_cost": "",
    "min_cost": ""
  }
}
```

Listing 4.3: JSON configuration file for parameters

4.3.2 DSE Framework

The DSE interface provides the control flow and the connectivity within all components of the DSE framework. It is also an interface between user-space (input JSON files and output reports) and the framework. It takes the application, the baseline hardware and set of parameters to start the DSE process. The interface class provides everything an algorithm needs: the design space (as flattened dictionary) a function to get results (from database if present, from simulation if necessary), a function to check whether one result is better than the other one and a function to check whether one result is sufficiently good to stop the simulation. In order to keep track on whether a simulation was already run it is assigned with a unique ID. Every time a result is requested from the DSE interface, a function "$track\_sim\_id$" is called. It checks whether the ID of the simulation configuration is already in the list of unique simulation runs. If so, it does nothing. If not, it adds the ID to the list and adds the time it took to run the simulation to the total execution time. The tools which are developed and integrated into the framework via DSE interface are as follow:

4.3.2.1 HDF5 (Storage) Handling

Provides functions to create hdf5-files (discussed in chapter 3.2.3) containing simulation results, to retrieve/add results from the hdf5-files and to flush & close hdf5-files. When a simulation returns an error and still reports some results (enough for DSE). Depending on the configuration, these results are stored in an extra file and will be reported to the user. This tool implements a function called "$hdf5merge.py$" to allow merging of failed results into a file with correct results. This leads to a complete data-set on which the framework may be run without the need for a new simulation. The "$hdf5merge.py$" also provides an option to replace the failed results with pre-defined values. The pre DSE simulation results, if they exist, can be given to the framework in the format of CSV/hdf5, the function "$csv2hdf$" converts results from the CSV-format to hdf5 based tables.

4.3.2.2 Cost Evaluation

This tool evaluates the architecture cost (decision metric) against the result of a new iteration of the given algorithm. It provides functions to calculate the EDP from the application execution time and the estimated power consumption of the SoC. It also provides a function to evaluate whether a certain configuration was already simulated or not. First, it tries to find the result in the given hdf5-table and if so, return the results (retrieved from the database). Only if the result is not in the table, the framework runs the simulation, adds the result to the table, and returns the value (also adds to the database) via the DSE interface.

4.3.2.3 Simulation Handling

This tool provides functions to easily generate simulation commands from the given configuration dictionaries and run these commands for the virtual platform simulation. The complete command to run the simulation executable binaries(.sparc), a pre-compiled PROM image (the boot code is used to initialize the run-time system and the peripherals), executable platform (eg. leon3mp.platform), SDRAM to use (if the simulation needs it), and intrinsics are extracted from user input JSON files. The output from the simulation and additionally specified scripts (e.g. power reporting) is passed to the next tool for extraction.

4.3.2.4 Virtual Platform Simulation, Value Extraction

Subsequently, the virtual platform is configured with the new hardware architecture parameters and loaded with the software. The resulting executable simulation model is then used to capture the previously described performance and power parameters. After the simulation, these parameters are summarized in a compact report file. The tool "$value\_extraction$" implements functions to extract values from the report. The extracted values, simulated time, total time and estimated power consumption are stored in hdf5 and used in DSE algorithms and user output reports.

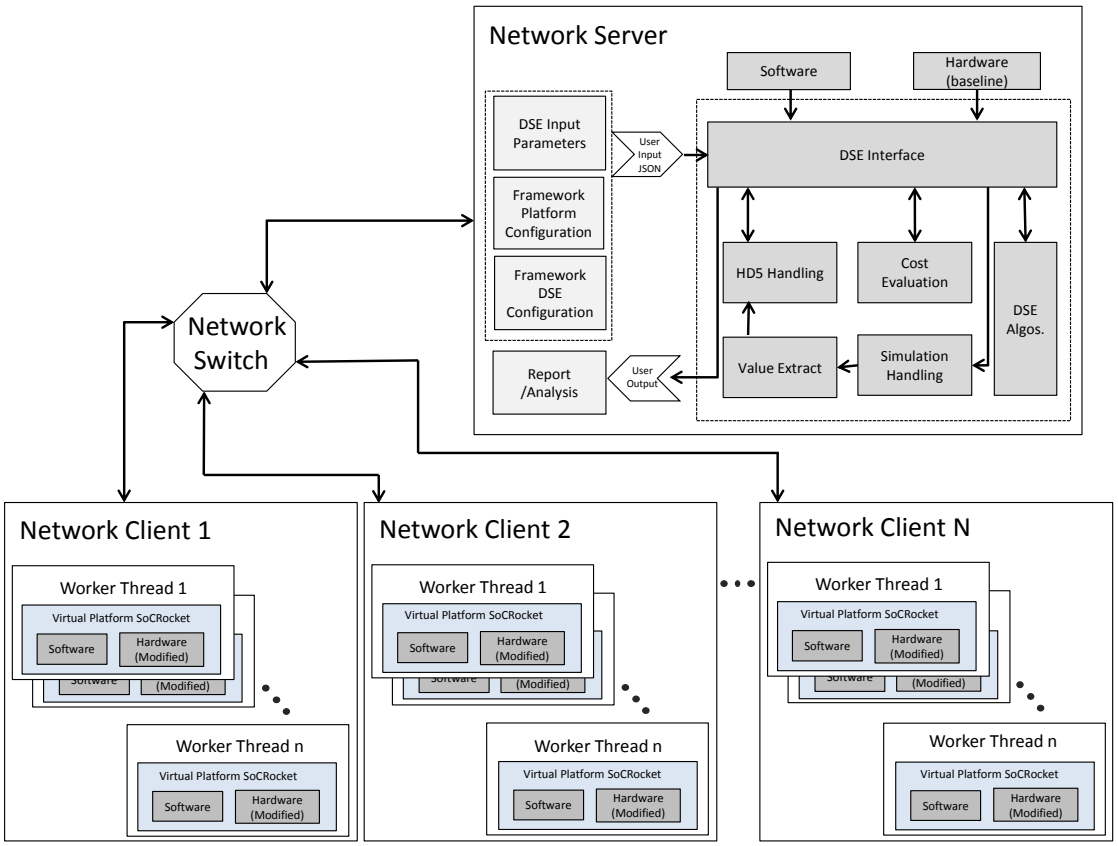


Figure 4.9: Automated Design Space Exploration Framework Network

4.3.3 User Output

Depending on the user selection of the algorithm, the DSE interface imports the algorithm to run DSE framework. It starts with an initial configuration (default, if it not is given by the user). The final output of the DSE framework, for any given algorithm, is in the following order. The best-found configuration, the corresponding best result (e.g. EDP), the number of iterations (algorithm convergence), whether the termination condition was met in finding ”sufficiently good result”, the number of unique simulations performed, and total time taken by the DSE process. All results are stored in hdf5 output files which are used in the final system-level analysis of SoC design.

4.3.4 Multi-Threaded Simulation Network

The parallel nature of the DSE strategies is suitable to exploit the speed-up advantage of multi-core distributed system, the presented parallel DSE framework (as shown in Figure 4.9) is capable of running simulations for the whole design-space and distributing this work across a number of multi-threaded systems (clients).

Network server: is a server when provided with the necessary configurations, can distribute work to available network clients running on the same network. For example, in a single iteration genetic algorithm, the number of individuals (configurations) in a population can be simulated concurrently. The network takes all the configuration parameters and passes to the clients running the virtual platform. In this way, during the DSE process all needed architectural configurations are simulated in the smallest amount of time by utilizing the maximum number of available systems.

Network client: is a worker program needed for the network server. Starting one worker thread from a multi-threaded client system for setting up the connection with the server using the address and the port of the server. This worker thread automatically requests the job from the server and then execute the necessary simulations. The client can be configured to use all but one available thread on a given machine, running several simulations in parallel. The network server provides an option (given by the user) to skip the fail simulation and run the next configuration. When this option is selected, the client is set to not cease working if a simulation fails as otherwise server and client may be stuck running the same failing simulation again and again. All simulation results are sent back to the network server to continue further the exploration process. The presented framework is tested on a network which has no problems regarding connection or firewall. If that is the case, a solution must be found (like disabling the firewall or creating an ssh-tunnel for the needed port).

4.4 Summary

This chapter presented an automatic and fast design space exploration framework. The DSE methodology was demonstrated with the simulated annealing and genetic algorithm-based optimization techniques. Application execution time and power dissipation of the

system was selected as the multi-objective of the DSE. Furthermore, all components of the presented DSE framework and parallel-simulations setup were described.

Chapter 5

Design Space Exploration Software

Today industry is rapidly transitioning from general purpose to more specialized approaches to achieve higher performance at a lower power dissipation of modern applications. To offload the computation intensive tasks from central process cores, several co-processor or accelerators are being integrated into upcoming SoCs. This co-designing trend brings serious engineering challenge for the engineers (system, hardware as well as software). It is becoming increasingly difficult to apply the traditional design flow of static hardware/software partitioning to early design stages. If the partitioning is performed without evaluating the achievable performance gain and cost then the design can lead to poor system resource utilization, subpar performance and retargeting of the architecture in later design stages. Recently industry and academia are addressing these design challenges by using virtual platforms and rapid prototyping frameworks based on SystemC/TLM2.0. These platforms are providing the designers with an executable prototype of the hardware well before any actual hardware sample becomes available. This enables the designers to start the development of hardware and software in parallel. However, exploration of hardware/software partitioning using these virtual platforms or hardware emulators remains a mostly manual and labor-intensive task. Developing a TLM based hardware representations for a section of software can be challenging for software developers. Herein lies the contribution of this work. This dissertation presents a workflow with a high degree of automation to explore different hardware/software partitions using a SystemC/TLM2.0 virtual platform. Based on sparse configuration input, the presented framework can generate a TLM based hardware accelerator for a given software function. It also annotates the accelerator with the timing and the power consumption parameters

derived from a high-level synthesis-based tool-flow. The next section will describe the scenarios for efficient exploration of hardware/software partitioning.

5.1 High-level Exploration of HW/SW Partitioning Scenarios

Improvements in power consumption and computational performance through technology scaling are diminishing which forces the system engineers to deviate from homogeneous architectures of the past. The transition into a heterogeneous age of computing is becoming increasingly popular [122]. Higher performance and lower power consumption now typically stem from specialized hardware accelerators tailored for specific workloads. Hence the traditional sequential development of hardware and software is breaking down, even independent development becomes challenging with the increased interdependence. This dramatically increases the responsibility of system designers for choosing the right partitioning between the two domains. In past the designer's experience was the major factor for the right design configuration selection, today's design space increasing at a rapid pace which requires new methodologies to reliably select the most suitable architecture.

This work addresses the continuously growing design space is composed of two components: intelligent strategies to parse the design space to reduce the number of necessary simulation-runs and hardware/software co-simulation (on a high abstraction level). The first component is already described in Chapter 4 whereas this chapter will illustrate the improvement of the second component; specifically, in supporting the designer to evaluate different hardware/software partitioning in a quick and easy fashion. When analyzing different hardware/software partitioning scenarios for a particular SoC, it is tempting to directly compare the raw computation performance of the general-purpose processor with the specialized hardware accelerator. However, this is a very narrow view of the hardware/software co-design process and its pitfalls. For accurate estimation the achievable performance gains, one must observe the complete interaction between accelerator, memory and software system (CPU) in its entirety [123]. This holds true for the analysis of both the performance and power/energy consumption of the system. The specialized accelerators or co-processors, by providing cache coherent access to the main memory from

different heterogeneous compute resources, are mostly employed in new and upcoming SoC architectures. The Coherent Accelerator Processor Interface (CAPI) developed by IBM [124] or the Heterogeneous System Architecture (HSA) [125] are examples for these architectures. While available, these coherent architectures not commonly used especially in the low power domain, as achieving cache coherency for multiple heterogeneous actors is costly in chip area and power consumption. These architectures are not within the scope of this work. Here the classic approach is investigated, where accelerators have their own fast memory without direct access to the main memory. The required data movement causes runtime overhead, reducing the achievable performance and increasing the power consumption. Equation 5.1 illustrates the individual components of the timing overhead incurred by the interaction between the accelerator, memory and software system. Here $t\_receive$ and $t\_transmit$ account for the time needed to move data to and from the accelerator. Additional time is needed to decode and encode the serialized data stream to be used by the computation and preparation of the data to be moved back to the main memory (t_decode and t_encode). The actual computation time required by the accelerator is represented by $t\_computation$.

$$t_{acc\_total} = t_{receive} + t_{decode} + t_{computation} + t_{encode} + t_{transmit} \tag{5.1}$$

Further elaboration on the accelerator generation workflow and its hardware/software integration into a virtual prototype platform follows below.

5.2 SW2TLM Design Flow

The date handling is the core part of the design flow of the SW2TLM. It can be achieved through different means; software and a DMA controller based method and their impact on the overall computation time are analyzed. These approaches are described in later sections in more detail. While the achievable performance is the central optimization goal, it cannot be viewed in isolation. A system's power and the resulting energy consumption are becoming increasingly important in modern SoC. SW2TLM includes early power estimation models based on concepts presented in Chapter 3. Power consumption is composed of two main components: static and dynamic power. These parameters are

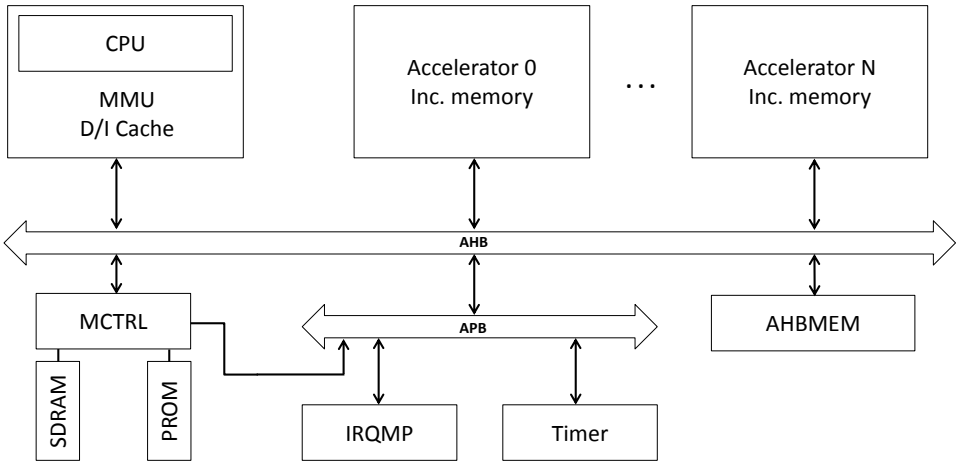


Figure 5.1: Baseline SoC architecture based on SoCRocket IPs

annotated to the accelerator hardware model and contribute to the overall power consumption estimated by the virtual platform used in SW2TLM. Following sections will illustrate the design space used for the architectural exploration, the individual tools comprising the automated framework and concluding with a look at the user interaction and usability of the framework.

5.2.1 System and Accelerator Architecture

Figure 5.1 shows the baseline architecture used for the design space exploration. It contains all necessary components to accurately reproduce the behavior of a complete SoC. The software is executed by a LEON3 instruction set simulator (Generated using TRAP [40]; see also section 2.3.2) that is embedded into an accurate MMU/cache architecture. Communication is handled via two buses AMBA High-performance Bus (AHB) and Advanced Peripheral Bus (APB) connecting the remaining components. In the figure, the investigated hardware accelerators are shown top right, they can be instantiated as AHB-Masters as well as slaves depending on the data transportation method. The explored architectures are based on fast local memories included in the accelerators itself. Consequentially input and output data must be moved to and from the accelerator. Data transportation is done by using two approaches, it is handled directly in the software (CPU) or handled by a DMA controller which is embedded within the accelerator to move the data independently.

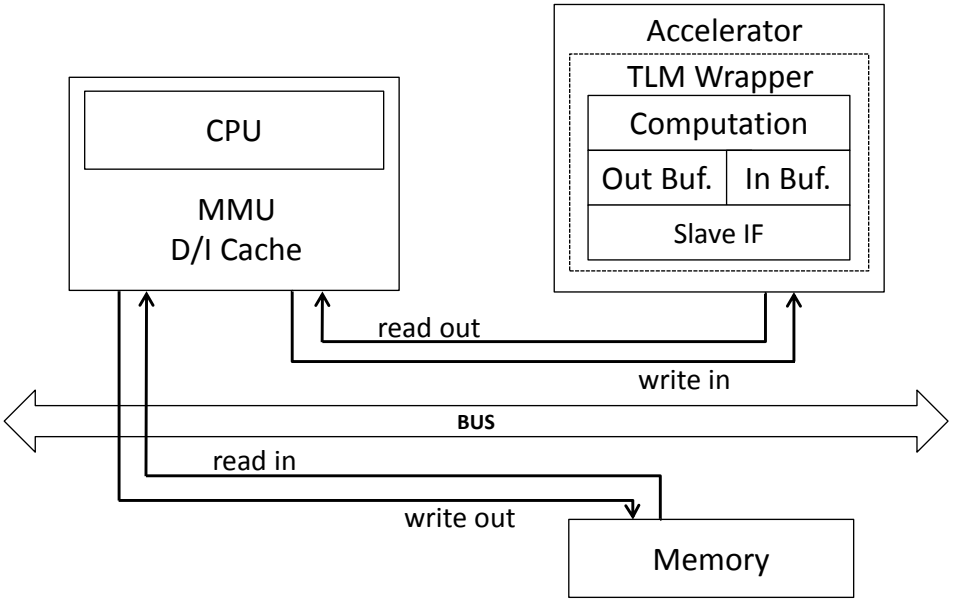


Figure 5.2: Accelerator architecture (Programmable I/O)

5.2.2 CPU Driven Data Transport (Programmable I/O)

Caches of a processor contain a copy of main memory when there are implemented by write-through policy. Using software data transfer approach, the data is moved directly from processor to accelerator. The structure of the hardware accelerator is shown in Figure 5.2, where the data handling is implemented directly in the software whereas the accelerator is used as an AHB-Slave with a simple blocking function call to perform the computation. Figure 5.3 illustrates a complete control flow for data transportation between processor and accelerator. The control flow is consisting of 4 steps:

- First, the length of input data is calculated in software and send to the accelerator, this data length later is used in the accelerator to ensure it received a complete data from the processor and no data is lost during communication.
- The actual input data required for the computation is transferred to the accelerator.
- After computation is finished, the processor sends synchronization (data-complete) signal to the accelerator.
- In the final step, the computed results from the accelerator are read by the processor.

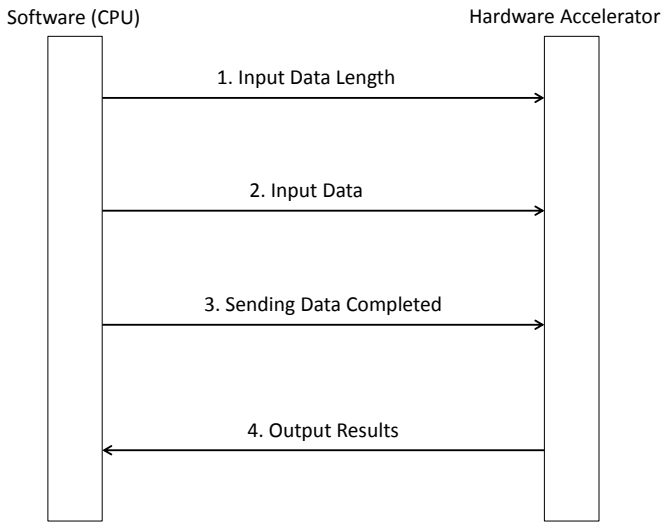


Figure 5.3: Data transfer control flow between CPU and accelerator (Programmable I/O)

Appropriate for the early exploration state, SW2TLM implements the accelerators' bus interface using the TLM2.0 LT coding style using blocking function calls. The specifics are illustrated below.

5.2.2.1 Implementation based on TLM blocking calls

In TLM nomenclature the CPU takes on the role of initiator whereas the accelerator is the target. Communication between the two is implemented in four stages shown in Figure 5.3 consisting of individual blocking function calls. As defined in the standard, a transaction's metadata (address, burst behavior, ...) and payload data is encoded into and passed as a generic payload object. Directly calling blocking functions from within the software execution by the ISS removes the need for explicit synchronization, as the initiator is blocked until the computation is completed. While this allows for a straightforward TLM implementation, software-driven transfer of large data sets is very inefficient, incurring a significant overhead compared to moving the data directly between main memory and accelerator local memory.

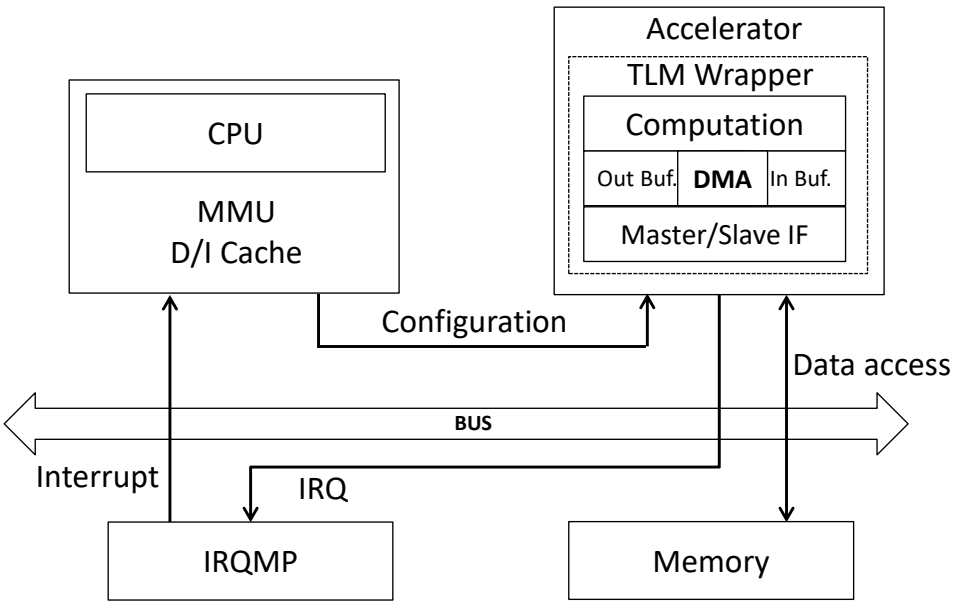


Figure 5.4: Accelerator architecture (with DMA Controller)

5.2.3 DMA Controller-based Data Transport

In order to reduce the CPU load and leverage efficient bus bursts, the accelerator was extended to include DMA functionality, independently reading and writing data to and from the main memory. To perform memory accesses on its own, the accelerator now is integrated as a master on the AHB bus and a slave on the APB bus (as shown in Figure 5.4). The AHB master interface is used to copy the input data from the main memory into an internal memory of the accelerator and writing back the computed resulting data from internal memory to the main memory. Whereas the APB interface holds three types of configuration registers described in Table 5.1, these are memory mapped registers and implemented with call-back functions in the TLM-wrapper. The implementation is described in the following section.

5.2.3.1 Implementation based on Memory Mapped Registers

The target hardware accelerator utilizes the SoCRocket memory mapped registers ($sr\_registers$) for DMA functionality. The instantiation and handling of $sr\_registers$ are identical to GreenReg registers. However, their usage results in a much smaller code

Name	Description
INDATA(i)[a]REG	Pointer to the input data (i) in the main memory
OUTDATA(j)[b]REG	Pointer to the output data (j) in the main memory
CONTROL_REG	Bit 0 = 1 starts the calculation

[a] i =1, ..., total number of input operands.
[b] j=1, ..., total number of output results.

Table 5.1: List of Configuration Registers

footprint which enables a simple mechanism for attaching callback functions to the registers. The SW2TLM provides $init\_registers$ function which creates the memory map registers for control, input, and output data. Moreover, it connects them to corresponding callback functions. Within the accelerator, these callback functions implement the data transfer functionality from memory addresses which are specified by the CPU. Listing 5.1 shows the implementation of $init\_registers$ function.

```
1  accelerator :: init_registers ()
2  {
3    r. create_register ("CONTROL", "ACC_CONTROL_REGISTER",
       CONTOL_ADDRESS_OFFSET, DEFAULT_VALUE, BUS_WRITE_MASK)
4     . callback (SR_POST_WRITE, this, &accelerator:: control_write );
5    r. create_register ("INDATA", " ACC_IN_DATA_POINTER",
       INDATA_ADDRESS_OFFSET, DEFAULT_VALUE, BUS_WRITE_MASK)
6     . callback (SR_POST_WRITE, this, &accelerator:: indata_write );
7    r. create_register ("INDATA", " ACC_OUT_DATA_POINTER",
       OUTDATA_ADDRESS_OFFSET, DEFAULT_VALUE, BUS_WRITE_MASK)
8     . callback (SR_POST_WRITE, this, &accelerator:: outdata_write );
9  }
```

Listing 5.1: Memory Mapped Registers Usage

These registers behave like an array and directly connected to the bus through bus_read and bus_write. Whenever the $sr\_register$ is written (SR_POST_WRITE), the corresponding callback functions $control\_write$, $indata\_write$ and $outdata\_write$ (see Listing 5.1 lines 4,6 and 8) are called. An example of callback function for CONTROL register is shown in Listing 5.2.

```
1 accelerator :: control_write ()
2 {
3   uint32_t ctrl = r[CONTROL];
4   unsigned int i = 0;
5   if ( ctrl && 0x1){
6     r[CONTROL] = i;
7     start_event . notify () ;
8   }
9 }
```

Listing 5.2: Callback function for CONTROL register Usage

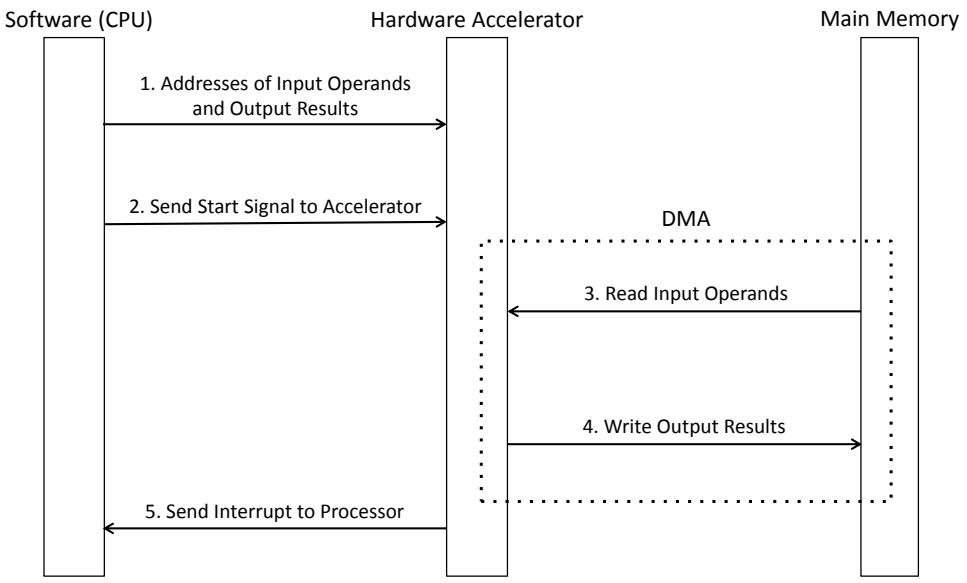


Figure 5.5: Data transfer control flow between CPU and accelerator (with DMA Controller)

Figure 5.5 depicts the data flow among the processor, accelerator and main memory. Here the software is only required to provide accelerator with the addresses of input operands variables and output variables which will store the accelerator computed results. An explicit synchronization between the processor and accelerator now must be implemented for DMA functionality. After the addresses (input/output variables) are written into data configuration registers, a signal is generated by writing '1' into control configuration

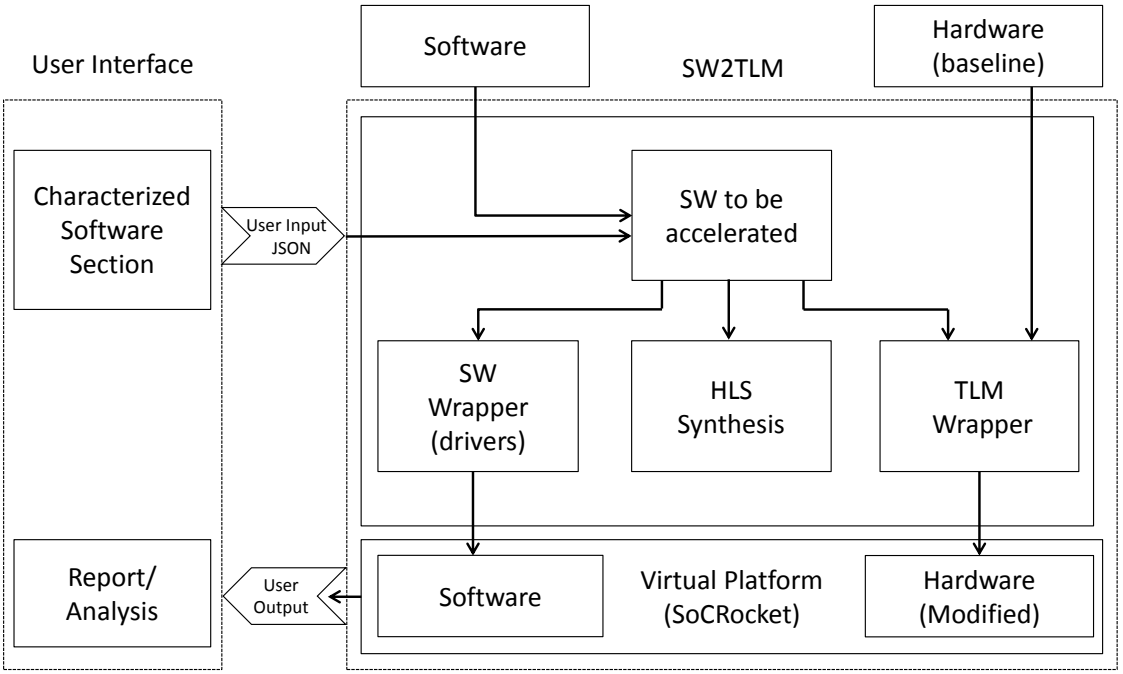


Figure 5.6: Baseline SoC architecture based on SoCRocket IPs

register to start the accelerator. Through the content (addresses) of the data configuration register, the accelerator reads input operands using AHB-Master interface from main memory. When the computation is done then the results are written back to main memory. As the architecture includes an interrupt controller (IRQMP), it is used to send an interrupt to the processor at the end of computation and data movement. Using accelerator with DMA reduces communication overhead and improves the overall performance, which is shown and discussed in Chapter 6.

5.3 SW2TLM Tool Flow

As stated before, the SW2TLM automation framework aims to hide as much complexity of the hardware/software co-design process from the user as possible. The structure of the framework is shown in Figure 5.6. For a given baseline design (hardware and software), the user only has to supply a characterization of the software sections to be accelerated in a compact and human-readable JSON file. Using this input, the framework generates hardware an accelerator for the specified software section to be simulated alongside the remaining software code by the virtual platform. Finally, comprehensive reports in-terms

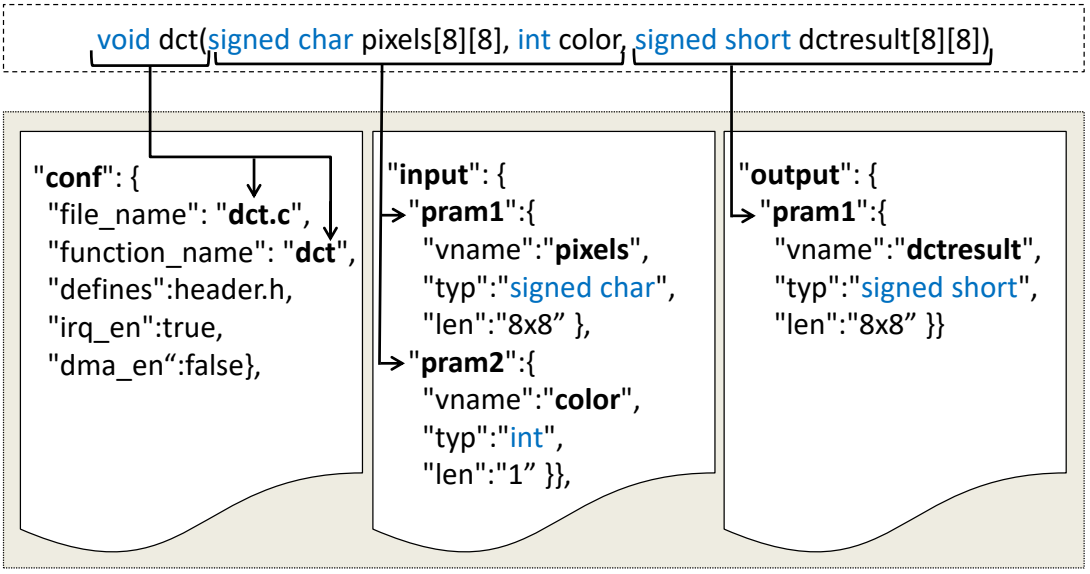


Figure 5.7: JSON based characterization of a software function

of performance and power consumption are generated which can be used to explore the design space in a bigger picture by including hardware/software partitioning. In the following sections, important components of the tool and integration flow are discussed in detail.

5.3.1 Characterization of the Software Sections

To show the high usability of the framework, an exemplary software characterization to be provided by the user is depicted in Figure 5.7. As shown in the three lower boxes of Figure 5.7, the contents of the JSON file are dived into three major categories: configuration, input, and output parameters.

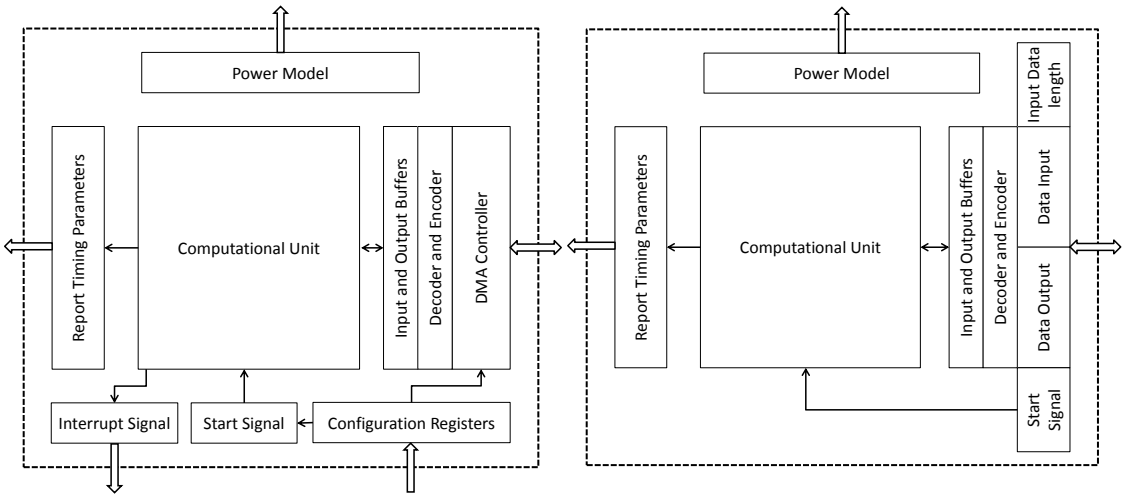
The configuration parameters Fields (”$file\_name$” and ”$function\_name$”) indicate the location of the source file storing the software function and the name of the function to be accelerated. If the software code has some user-defined data-types e.g structures, then the location of the file contenting its definition must also be given in the field ”defines”. The configuration parameters also include the means of synchronization in the Field ”irq_en” to indicates that the processor is notified via interrupt after the accelerator has finished the computation and the type of data transportation in the Field ”dma_en”, the user can

select the memory management applied for the accelerator. Enabling the DMA will include a dedicated controller in the accelerator, to read and write from the main-memory independently (see Figure 5.2). If disabled, the software wrapper is extended with additional code to transport input and output data to and from the accelerator (see Figure 5.4). The fields of input and output parameters differentiate the variables into inputs and outputs and the annotation of type and length which are used to generate decode and encode functions needed to serialize and de-serialize the data.

5.3.2 Annotation of Power and Timing Information into Hardware Model

In order to get the comparable results for design exploration, the software functionality (which needed to be accelerated) must be part of a suitable hardware model. This requires annotation of realistic performance parameters: computational delay and power consumption numbers of corresponding hardware implementation. The SW2TLM framework uses state-of-the-art High-level synthesis (HLS) and RTL synthesis tools (like Xilinx Vivado-HLS and Synopsys design compiler) to generate these performance parameters.

Timing information is extracted from the HLS synthesis reports and then it is annotated within the system-level hardware model (without directly interfacing the generated RTL hardware model with the virtual platform) to reproduce realistic computational delay. Similarly, extracted power numbers from synthesis reports are used to create a new power model for the accelerator, the model is integrated into the overall power estimation framework as discussed in Chapter 3. When the RTL model is directly used in the virtual platform, then not only the simulation performance would severely degrade but it also requires a complex transactor implementation to bridge clocked (RTL) and un-clocked (TLM) sections of the investigated hardware architecture.



(a) Accelerator (with DMA Controller) (b) Accelerator (Programmable I/O)

Figure 5.8: SW2TLM generated TLM wrappers

5.3.3 TLM Wrapper

Using the input of JSON file, the framework determines the type of accelerator and generates the corresponding TLM wrapper, the computational part of the function to be accelerated is extracted from the baseline software and encapsulated as a computational unit within the TLM wrapper as shown in Figure 5.8. This wrapper provides the functionality needed for communication and synchronization between the accelerator and processor. While the computational unit, power/performance reporting, and encoder and decoder logic are similar for both wrapper types, configuration and data movement components need some further elaboration. If a wrapper with DMA is configured, the internal structure depicted in Figure 5.8a is generated. Here the DMA controller performs direct accesses (via standard SoCRocket function calls; $ahbread$ and $ahbwrite$) to the memory via its bus master socket. Instructions for the controller are configured through the configuration registers, receiving input and output data addresses and the corresponding length. Data retrieval and computation are similarly started by writing to the start signal register, while an interrupt is triggered after the resulting data was written back to the main memory. Thereby notifying the ISS, that normal execution can continue. The wrapper without DMA observes a simpler internal structure (see Figure 5.8b). Instead of a dedicated controller with its own control flow, everything is executed within the blocking function call

of the ISS. As all results directly become available after the blocking function call returns, the software can directly read back the output data without any explicit wait states or synchronization via status registers or interrupt signals.

5.3.4 Software Wrapper

To integrate the accelerator into the control flow of the reference software, a corresponding software wrapper is generated to facilitate the communication with the accelerator hardware. This entails data encoding, decoding, the movement to and from the accelerator as well as synchronization after the computation is completed. The Listing 5.3 shows a sample software wrapper to handle data transportation (Lines 6,8 and 12) and communication (Line 10) between processor and accelerator (Programmable I/O, as described in Section 5.2.2).

```
1  unsigned char * hw_acc_addr;
2  hw_acc_addr = 0xb0000000;
3  unsigned int  hw_acc_addr_data_offset  = 0x10;
4  unsigned int  hw_acc_addr_complete_offset  = 0x4;
5  // Step 1: sending data-length to accelerator
6  *((unsigned int *)hw_acc_addr) = sizeof( input_data );
7  // Step 2: sending data to accelerator
8  memcpy(hw_acc_addr+hw_acc_addr_data_offset, input_data , sizeof( input_data );
9  // Step 3: signaling end of data movement
10 *(hw_acc_addr + hw_acc_addr_complete_offset ) = 1;
11 // Step 1: receiving results from accelerator
12 memcpy(result,hw_acc_addr+ hw_acc_addr_data_offset , sizeof( result ));
```

Listing 5.3: Software wrapper for data transportation

An example of the software wrapper for accelerator with DMA controller (discussed in Section 5.2.3) is shown in Listing 5.4. The SW wrapper is also responsible for initializing (Line 5) the interrupt controller and setting up the interrupt handler "HW_ACC_IRQ" flag for the CPU. Addresses pointing to the input and output variables (Lines 7-9) are

passed to the accelerator. Line 11 shows the value of control-register is changed (from 0 to 1), starting the accelerator. Lastly, after computation and data movement is completed, an interrupt is emitted to signal the CPU. This interrupt is caught in the service routine and the previous software execution is unblocked by writing to the corresponding flag (Line 13).

```
1  unsigned int * CONTROL_REG = (unsigned int *)0x80000b00;
2  unsigned int * INDATA1_ADDRESS_REG = (unsigned int *)0x80000b04;
3  unsigned int * INDATA2_ADDRESS_REG = (unsigned int *)0x80000b08;
4  unsigned int * OUTDATA1_ADDRESS_REG = (unsigned int *)0x80000b0c;
5   irq_init (HW_ACC_IRQ);
6  // Step 1. write addresses of input/output variable to data configuration
       registers
7  *INDATA1_ADDRESS_REG = (unsigned int)&inputdata1;
8  *INDATA2_ADDRESS_REG = (unsigned int)&inputdata2;
9  *OUTDATA1_ADDRESS_REG = (unsigned int)&outputdata1;
10 // Step 2. issue signal for starting accelerator
11 *CONTROL_REG = 1;
12    // Step 5. wait for an interrupt from the accelerator
13  irq_wait (HW_ACC_IRQ)
```

Listing 5.4: Software wrapper for data transportation

5.3.5 Performance and Power Parameters Summary

Table 5.2 summarizes the timing parameters used for the performance comparison of the system with and without hardware accelerator. Most of the parameters were already defined in Section 5.1. Further parameters are: Total communication overhead introduced by the accelerator ($t_{com\_overhead}$), execution time of the characterized software section running on the ISS (t_{sw}) and the absolute application runtime independent of the chosen implementation (t_{app}).

Parameter Name	Description
$t_{receive}$	Writing data to HW ACC directly or via DMA
$t_{transmit}$	Reading data from HW ACC directly or via DMA
t_{decode}	Decode data within HW ACC
t_{encode}	Encode data within HW ACC
$t_{computation}$	HW ACC computation time
$t_{com\_overhead}$	Communication overhead
t_{acc_total}	Total HW ACC execution time
t_{sw}	Execution time in of selected software section
t_{app}	Absolute application execution time

Table 5.2: Timing parameters evaluated in the performance analysis

Parameter Name	Description
p_{Static}	Static power consumption
$p_{internal}$	Internal part of dynamic power consumption
$p_{switching}$	Switching part of dynamic power consumption
p_{total}	Total power consumption
e_{total}	Total energy consumption

Table 5.3: List of investigated power and energy consumption parameters

The investigated parameters regarding the power consumption are listed in Table 5.3. Static (p_{Static}) plus dynamic ($p_{internal} + p_{switching}$) is used to calculate the total power consumption of a specific hardware component as well as the complete platform. These parameters are represented as the average power draw, in addition the consumed energy until task completion is calculated by Equation 5.2.

$$e_{total} = p_{total} + t_{total} \tag{5.2}$$

5.3.6 Platform Integration of the Accelerator and System Simulation

Finally, the virtual platform is configured to instantiate the generated accelerator transaction level model. The framework connects all accelerator ports to the corresponding system ports e.g. DMA controller ports are connected to AHB bus and accelerator configuration registers are connected to the APB bus, similarly the interrupt port of accelerator

is connected IRQMP port. Subsequently, the modified software containing the new function implementation without actual is wrapped into the SW wrapper and loaded into the virtual platform. The resulting virtual platform is simulated to capture the previously described performance and power parameters of the newly partitioned execution platform. After the simulation is done, the results are collected and summarized in a compact and humanly readable report file.

5.3.7 Summary

In this chapter central characteristics and workflow of the SE2TLM framework were presented. It automates functionality ranging from power and performance characteristics estimation of software code blocks using state-of-the-art HLS to the seamless integration of a dedicated accelerator into the hardware and software environment of a virtual platform. Next to raw performance and power data, additional attention is allocated to the communication between software and hardware. For which the framework allows two distinct modes based using direct memory access and programmed IO. In total SW2TLM enables designer without deep architecture knowledge to explore the benefits of different partitioning scenarios. A detailed evaluation of these features for a set of real-world applications is documented in the following chapter.

Chapter 6

Evaluation Results

Required by the strong inter-linked nature of the problem domain addressed by the presented tools, evaluation is performed using a set of common real-world use cases. They include computation and data-intensive applications, making them ideal candidates to exploit the parallelism and custom hardware acceleration. At the core, addressed application domains are typically resource constrained and require a configuration with high performance while limiting the overall system's cost in terms of silicon area and power consumption. Addressing these optimization goals, the presented research framework is evaluated towards reducing design space exploration time using simulated annealing and genetic algorithms. Discussing the greatly reduced number of required simulation-runs and parameter selection accuracy. Extending on this, the framework features aiding hardware/software-partitioning are evaluated using a subset of the presented applications. Here the evaluation focusses on custom accelerator performance gains, incurred communication overhead, and reduction of overall power consumption.

6.1 Use Cases

Image processing was selected as the main test case domain in this work, it is resources hungry and requires real-time computation. Typical examples of such intensive computation operation are compression, convolution, segmentation etc. These operations often perform intensive matrix multiplications over all pixels of an image, which needs significant resources of the SoC in terms of computation, data bandwidth and memory. Hence

making it difficult for a system design engineer to meet the functional and non-functional constraints. If the under-laying system architecture is not selected properly, application performance may degrade significantly. SoC used in space satellite are common examples of resource constraint system, where the area and power are critical. In this work, Leon3mp is chosen as the base system, it is widely used in different projects of the European Space Agency (ESA). The base platform is optimized for the execution of different image processing algorithms and test benches. These algorithms are defined in the following sections.

6.1.1 Image Filtering

Image filtering is the process that translates an input image by applying a certain mask (also known as filter or kernel is a small matrix, typically 3x3 or 5x5) to an output image. The resultant output filtered image is a linear combination of the neighboring pixels in the input image. Equations 6.1 and 6.2 shows the value of an output pixel, corresponding to a input pixel located at (x, y) in a 2-dimensional image, obtained through a weighted summation of input pixel and its surrounding pixels. Equation 6.1 shows a cross-correlation operation which applies the filter directly on the image whereas the convolution operation (see equation 6.2) first flipped the filter (both horizontally and vertically) and then apply it to the image.

$$I_{out}(x, y) = \sum_{i,j} f(i, j) + I_{in}(x + i, y + j) \tag{6.1}$$

$$I_{out}(x, y) = \sum_{i,j} f(i, j) + I_{in}(x - i, y - j) \tag{6.2}$$

Image filtering is commonly used for noise removal and contrast enhancement in the image. Computer vision applications use filtering to extract useful information such as edges and corners for pattern detection and template matching. Sobel which is an approximation of Gaussian derivatives, median and box filters were implemented in the scope of this work.

6.1.2 Integral Image

Often called summed-area table, the idea originally proposed in 1984 for image texture mapping[126]. Later, Viola and Jones in [127] first time introduced this idea of using an integral image for computer vision into their face detector. Calculation of integral image is shown in equation 6.3, an output image pixel at location (x, y) is summation of all pixels left $(\acute{x} \leq x)$ and above $(\acute{x} \leq y)$ of the input image pixel. This algorithm is used in many computer vision applications such as Speeded-Up Robust Features (SURF)[128] and Fast Approximated Scale-invariant feature transform (SIFT)[129] for implementing image pyramids, also used in local tone mapping for high dynamic range (HDR) rendering [130].

$$I_{out}(x, y) = \sum_{\acute{x} \leq x, \acute{y} \leq x} I_{in}(\acute{x}, \acute{y}) \tag{6.3}$$

6.1.3 CCSDS123

It is an algorithm for lossless compression of hyperspectral images [131] as standardized by the Consultative Committee for Space Data Systems (CCSDS) in Standard #123 [132]. It is an example of the applications executed on the payload processors aboard scientific satellites. Hyperspectral images are three denominational data sets: two dimensions are spatial and third is spectral. A hyperspectral image is considered as a stack of individual frames representing the same scene in different regions of the electromagnetic spectrum. The algorithm based on adaptive linear predictive compression, which uses the algorithm with local mean estimation and subtraction for filter coefficients adaptation. The prediction residual is finally encoded with a sample-adaptive Golomb-Rice encoder.

6.1.4 Hotspot

This is a benchmark [133] based on a thermal simulation tool ($HotSpot$) which uses an architectural floor plan and simulated power measurements for estimating the processor temperature. The benchmark takes power and initial temperatures as input and through iterative solutions of a series of differential equations gives different output cells that represent the average temperature value of the corresponding area of the chip.

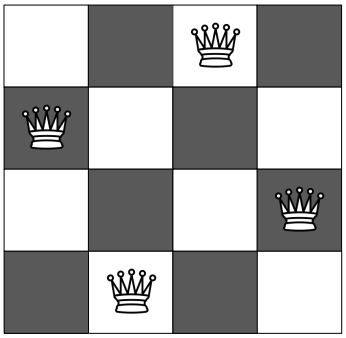


Figure 6.1: N queen placement puzzle

6.1.5 Nqueen Puzzle

It is an algorithm to solve the N queen problem which is to find the valid placement (an example is shown in figure 6.1) of N queens on a NxN chessboard, so that no two queens attack each other. In simple words: no queen shares a row, column or diagonal paths with other queens.

6.1.6 JPEG Codec

JPEG Codec is a widely used lossy image compression format. The main parts of a JPEG encoder are Color Space Conversion, Chromatic Sub-Sampling, Block Splitting, Discrete Cosine Transformation (DCT), Quantization and Entropy encoding. DCT is a baseline foundation of JPEG compression it includes complex mathematical operations such as Fast Furrier Transforms (FFT) which requires massive computations. In this work DCT functionality selected for hardware/software exploration, SW2TLM automatically generates a corresponding hardware accelerator of it while the remaining functionality JPEG compression stays in the software on the host CPU.

6.2 Implementation of Use Cases

The above-described set of use cases were implemented in C language for DSE and SW2TLM evaluation. The applications are executed as bare-metal software for single core DSE and SW2TLM. For the multi-core DSE, they were run by a single master CPU

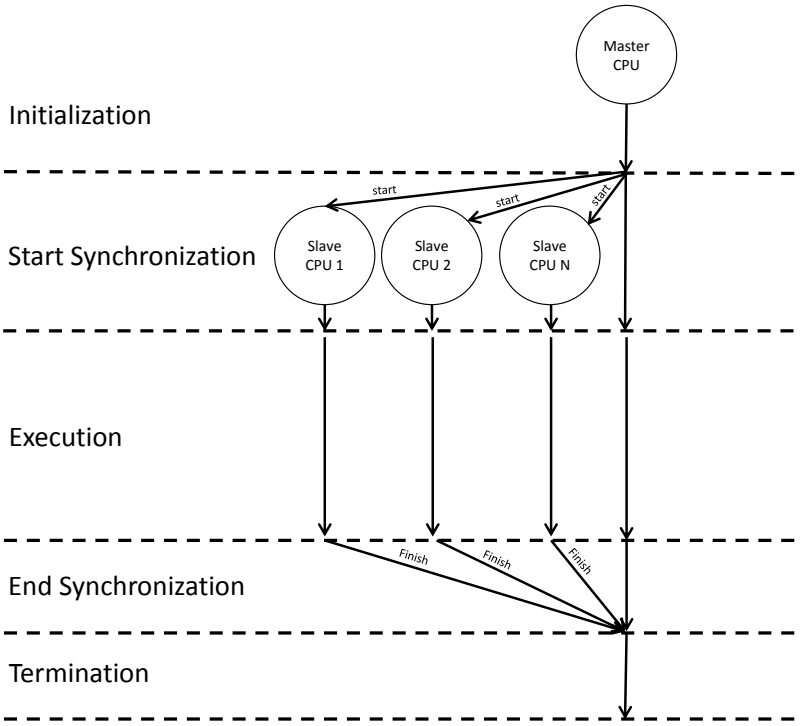


Figure 6.2: Single master CPU - multi salve CPUs implementation

- multi salve CPUs configuration illustrated in the following section. Sophisticated parallel optimization techniques (e.g. complex programming models) of many-core systems are not in the scope of this dissertation. Therefore, they were not applied for evaluation purposes.

6.2.1 RTEMS-based Multi-cores Implementation

The master CPU runs Real-Time Executive for Multiprocessor Systems (RTEMS) [134], which is an open source operating system with standard API support. RTEMS is most widely used in real-time applications such as space flight. It is supported by numerous processors architectures such ARM, Texas Instruments, Atmel AVL and most commonly used in the development of multiple microprocessors system (e.g. SPARC, ERC32, LEON, PowerPC and many more) for space domain. In this work, the applications were mapped to the Leon3 multi $(1, 2, 4, 6)$-processor system, Figure 6.2 shows the implemented design flow of the single master CPU and multi-slave CPUs system.

- **Initialization phase**: an RTEMS task is created for single master CPU execution whereas other slave CPUs run bare-metal. During this phase, all salves processors are blocked until all input data is read and all global data structures of the system, stored in shared memory, are initialized. An example of control type mp_data, a shared data structure for the integral image use case, is given in Listing 6.1. The data structure is intended for global distributed control and synchronization (barrier_t and lock_t); additionally, it also contains input and output data buffers.

- **Start Synchronization Phase:** a barrier (barrier_t) for thread synchronization and a global locking (lock_t) to maintain mutual exclusion (prevent simultaneous data writing and reading) are implemented. After system boot up and system initialization is done, the master signals (start) to all waiting salve processors.

- **Execution Phase:** once initialization sequence and synchronization phase are completed. Then, each processor starts computation by acquiring data block. The processors then independently execute the code encapsulated in the compute function. In the case of shared data accesses, the data block is protected by a lock to make sure that only one processor may acquire the block at a time.

```
1  typedef struct mp_data{
2       barrier_t   barrier ;
3       lock_t  lock;
4       int  image[width*height ];
5       int  newimage[width*height];
6    } mp_data_t;
7    extern mp_data_t *mp_data;
```

Listing 6.1: Example shared global data structure

- **End Synchronization Phase:** all processors need to synchronize at the end of the simulation, all slave processors have to be completed before the master is shut down. The master waits for signals (finish) from every processor, every processor is allowed to run ahead until its computation is completed.

- **Termination Phase:** the master CPU is responsible for collecting and assembling the computed results generated by all processors. Finally, the results are stored/steamed out and simulation is terminated.

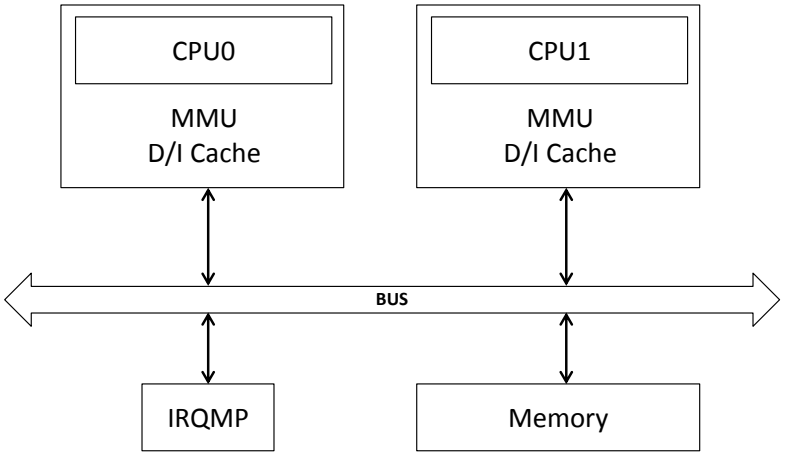


Figure 6.3: Example of two cores systems

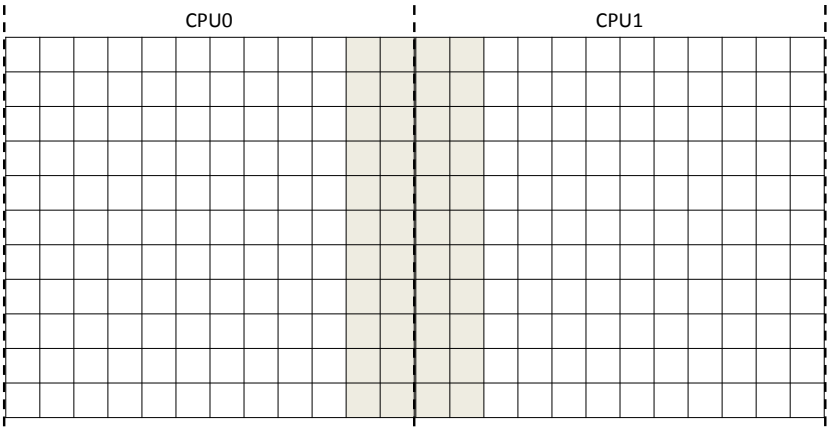


Figure 6.4: Workload distribution two cores system

6.2.1.1 Example of 2 Cores System:

An example multi-core system with 2 CPUs, shared bus and memory is shown in Figure 6.3, the target application is image filter with a 3x3 kernel. CPU0 runs RTEMS task which boots up and initializes the system, it specifies a shared data structure in shared memory to store the kernel. CPU0 reads image and stores to the memory. After synchronization, both CPUs start actual computation. An example of a workload is illustrated in Figure

Parameter Name	**Set**	**Set Values**
Number of Processors (nCPU)	p_1	1,2,4,6 (on shared bus)
Instruction Cache Banks (nICS)	p_2	1,2,3,4 (per processor)
Instruction Cache Bank Size (ICSsize)	p_3	1,2,4,8,16 KB
Data Cache Banks (nDCS)	p_4	1,2,3,4 (per processor)
Data Cache Bank Size (DCSsize)	p_5	1,2,4,8 KB

Table 6.1: Optimization Parameters

6.4. In this example, an input image of size 24x11 pixels used for filtering. The input image is virtually divided into two (12x11 pixel) segments, CPU0 processes left segment, whereas CPU1 processes right segment of the image. Grey color in the figure shows the overlapping area of the image between two CPUs to take care of the inter-segment boundary condition. Both CPUs store their computation in the output buffers. Finally, CPU0 write the filtered image to an output device. In this research different, selected optimization parameters cache configurations and the number of processors, which can be seen in Table 6.1, are explored for multi and single core implementation. The base platform is optimized for executing the set of image processing applications, described in the previous section.

6.2.2 Bare-metal Implementation

The use cases were ported to run on a single bare-metal $leon3mp$ platform. Figure 6.5 shows an example of JPEG compression on a bare-metal system for the SW2TLM framework. The input device reads a BMP input image either from an external memory or camera image stream (not implemented at time of writing, it requires a camera image signal interface which is out of scope for this work). Input image data is stored in SRAM where it can be accessed by the CPU and hardware accelerator. The whole JPEG compression application is mapped to the CPU and the DCT computation is offloaded to a hardware accelerator. Results are stored in SRAM where it is can be read by an output device. For the purposes of this work, it is streamed into an output file, but it can be modified to stream the images to an external display.

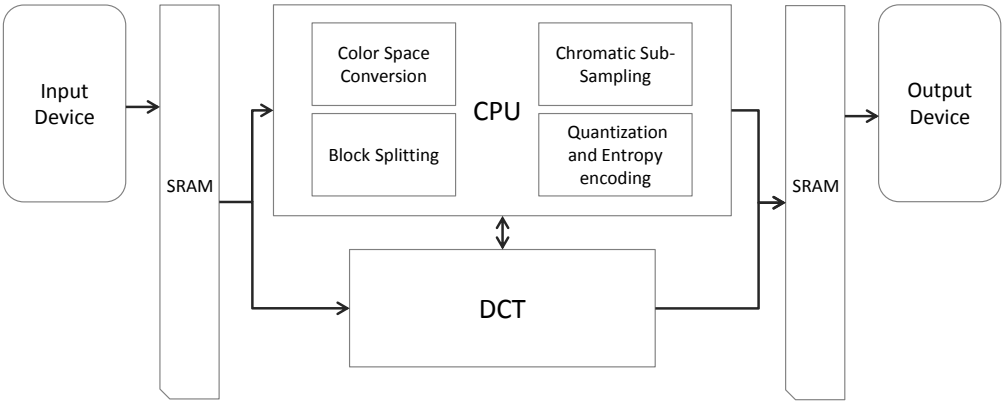


Figure 6.5: JPEG Compression on bare-metal system

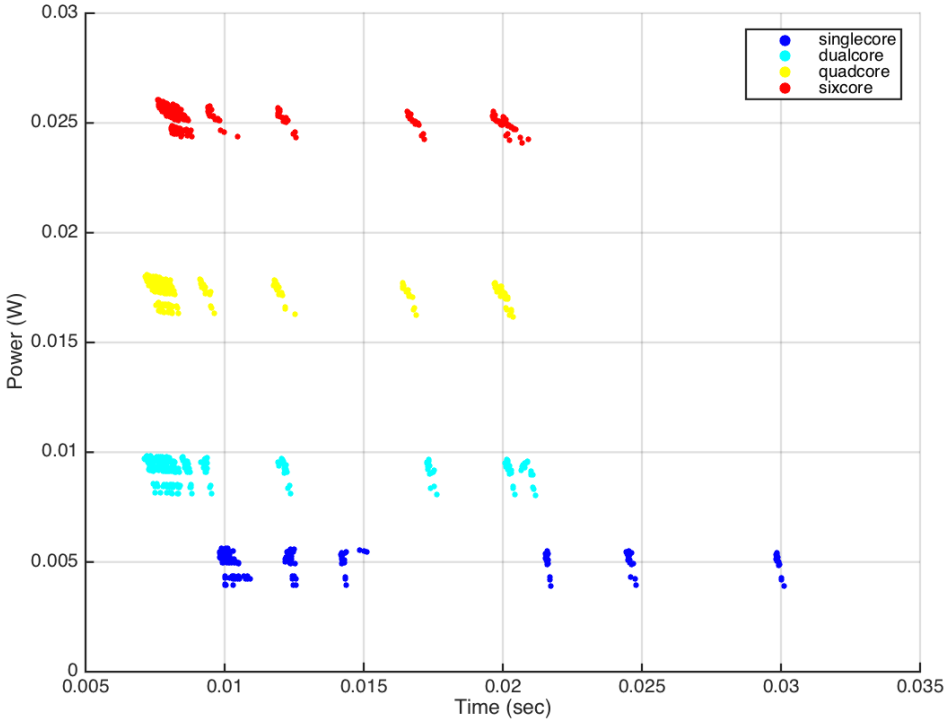


Figure 6.6: Multi-core System DSE: Integral Image

6.3 DSE Framework Performance Analysis of Simulated Annealing-based Optimizations

The implemented use case demands high performance within a limited power budget to meet real-time and efficiency requirements. The performance-power-consumption trade-off for the applications is illustrated in Figure 6.6. For purposes of brevity, only the

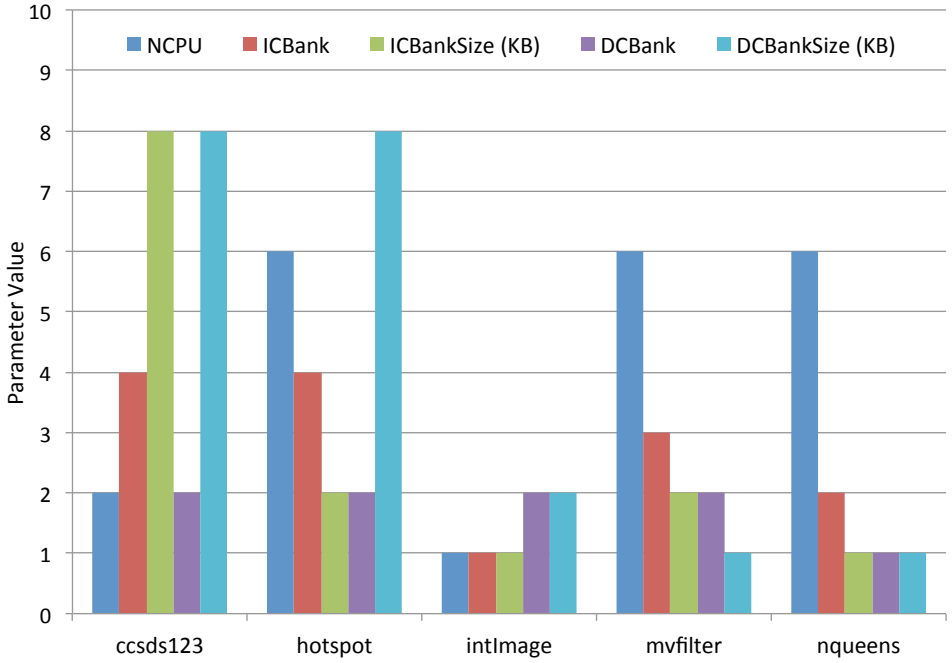


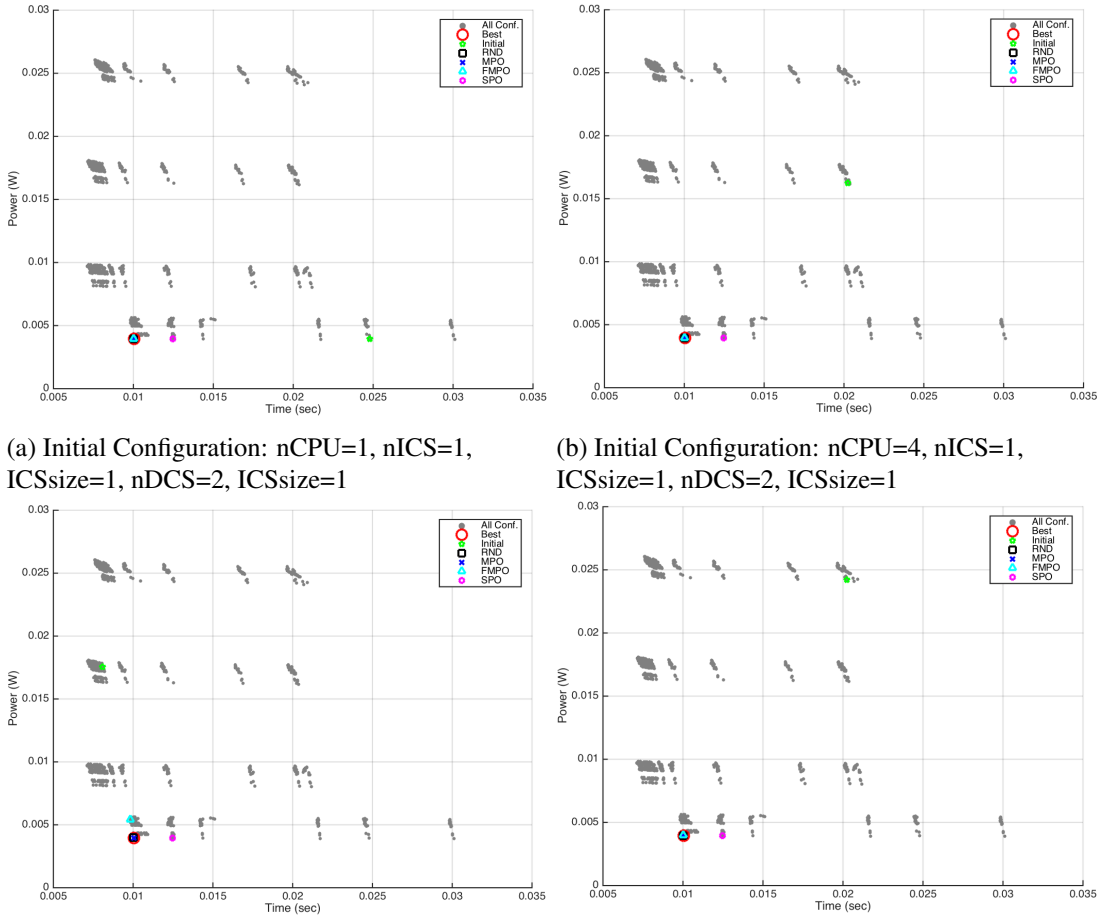
Figure 6.7: Multi-core: optimal parameters configuration with best application's performance

Integral Image is shown, the remaining results for the other implemented applications are given in the Appendix A.

The design space shown in the figure spans all parameters indicated in Table 6.1 (for better visual perception only the number of cores are highlighted in the graph) based on simulated system time and average power consumption, which are the two components of the EDP metric. The results of the exhaustive search with the best performance in terms of minimum EDP are depicted in Figure 6.7. It can be seen, that the instruction cache has a big impact on the performance. However, due to high data locality in the code, the impact of the data cache is rather low. Compared to the single-core system, multi-processor systems require larger cache sizes to leverage high performance.

The individually selected optima by the DSE algorithms is shown in Figure 6.8. The initial parameters configuration is highlighted with the green star and target parameters configuration (results of ES) is shown as a red circle. Four different initial parameters configurations are shown in the sub figures and every DSE framework algorithm starts with the same initial configuration for given subfigure. It visible from the figure, independent of initial start the framework convergences to the optimal configuration. The

DSE framework results are comprehensively analyzed in the following sections with regards to speed and accuracy. All simulation results are gathered using the OSCI SystemC reference simulator on a simulation server (4x Quad XEON, 3.4 GHz, 32GB RAM).



(a) Initial Configuration: nCPU=1, nICS=1, ICSsize=1, nDCS=2, ICSsize=1

(b) Initial Configuration: nCPU=4, nICS=1, ICSsize=1, nDCS=2, ICSsize=1

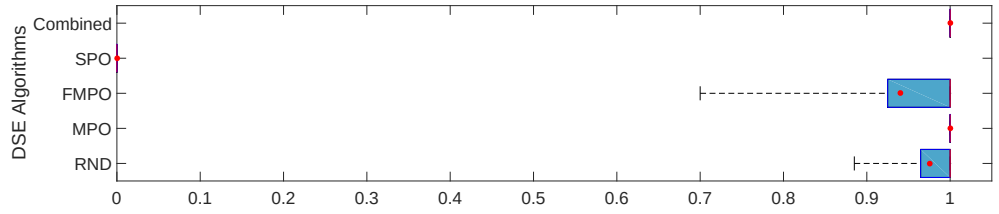
(c) Initial Configuration: nCPU=4, nICS=4, ICSsize=2, nDCS=2, ICSsize=2

(d) Initial Configuration: nCPU=6, nICS=1, ICSsize=1, nDCS=2, ICSsize=1

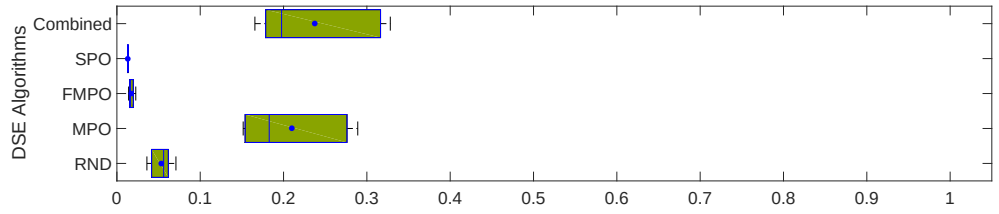
Figure 6.8: Multi-core System DSE Framework Performance: Integral Image

6.3.1 DSE Performance Comparison: Multi-cores System

The DSE algorithms performance in terms of accuracy (desirable or minimum EDP selection) and speed (the number simulations performed) is compared to an exhaustive search (ES) which was carried out to find the optimal baseline system. To find the optimal parameters configuration, every optimization algorithm runs one time with the exception of the RND algorithm where it is an average of 10 runs (to minimize randomness



(a) Selection Accuracy (Normalized to ES)



(b) Number of Simulations (Normalized to ES)

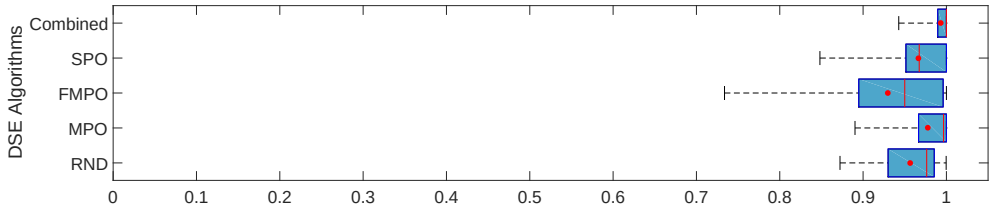
Figure 6.9: Multi-Processor System DSE

in the RND algorithm). Figure 6.9[1] shows the evaluation of each algorithm, where the selection accuracy is represented on the X-axis as the percentage of ES's EDP achieved, while the percentage of ES's simulations performed (also on X-axis) as the number of simulations of each algorithm. The median is shown by a vertical line and average drawn as a dot for each algorithm.

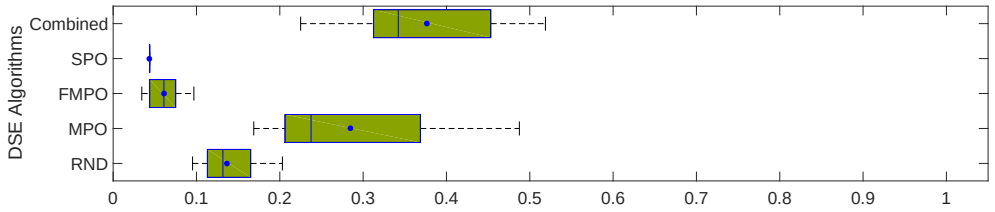
The results of RND show, that on average with 5.4% of ES simulations the selection of 97% minimum EDP is achieved. The SPO algorithm is worst in multi-processor DSE and failed to find the optimal parameters configuration with desirable EDP. The main reason behind this is that the multi-processors system shows a higher performance with larger caches which overcome the bottleneck of a shared bus. With MPO, we achieved a 100% of minimum EDP selection accuracy. However, it has a relatively higher number of simulations. On an average 21% of ES's simulations are needed to find optimal parameters. The $FMPO$ algorithm is the best trade-off between selection accuracy and speed of the optimization algorithm. Its results show a selection accuracy of average 94% and number of simulations of below 2% as compared to ES.

It is important to note that in practice a lookup table is maintained for the configurations already simulated. This allows the framework to reuse the results for simulations when running several algorithms together. Therefore, the total number of simulations required for all algorithms together is less than the sum of simulations required for algorithms

[1](Performance of DSE Algorithms: For selection accuracy higher is better and number of simulations lower is better)



(a) Selection Accuracy (Normalized to ES)



(b) Number of Simulations (Normalized to ES)

Figure 6.10: Single Processor System DSE

separately. This means the approach "$Combined$", which would have seemed quite expensive is actually only slightly worse than the MPO algorithm in terms of the number of simulations required (24% of ES's simulations). However, it is significantly better in achieving the minimum EDP (100% of ES's EDP).

6.3.1.1 Single Core System DSE

The bare-metal MiBench [135] and several other open source test benches are used for a single processor system DSE. Each test is designed to run on a single $leon3mp$ platform and is executed with the different cache configurations. The best parameters configuration of each test with the best performance is shown in the Figure 6.11, which constitute the target goal of the DSE framework. Similarly to the multi-processor system Figure 6.10[2] shows the percentage of ES's EDP selection and percentage of ES's simulations needed to get optimal parameters. The figure demonstrates the average of 14% simulations and the selection accuracy of 95% with the results of RND is achieved. For the single-processor system, the SPO and $FMPO$ are the fastest algorithm with an average of 5% and 6% of ES's simulations respectively. The SPO selected 96% of desirable EDP, whereas $FMPO$ achieved 92% of desirable EDP selection. The MPO achieved a high selection accuracy (average 97% ES's EDP) while needing 29% of the simulations

[2]Performance of DSE Algorithms: For selection accuracy higher is better and number of simulations lower is better

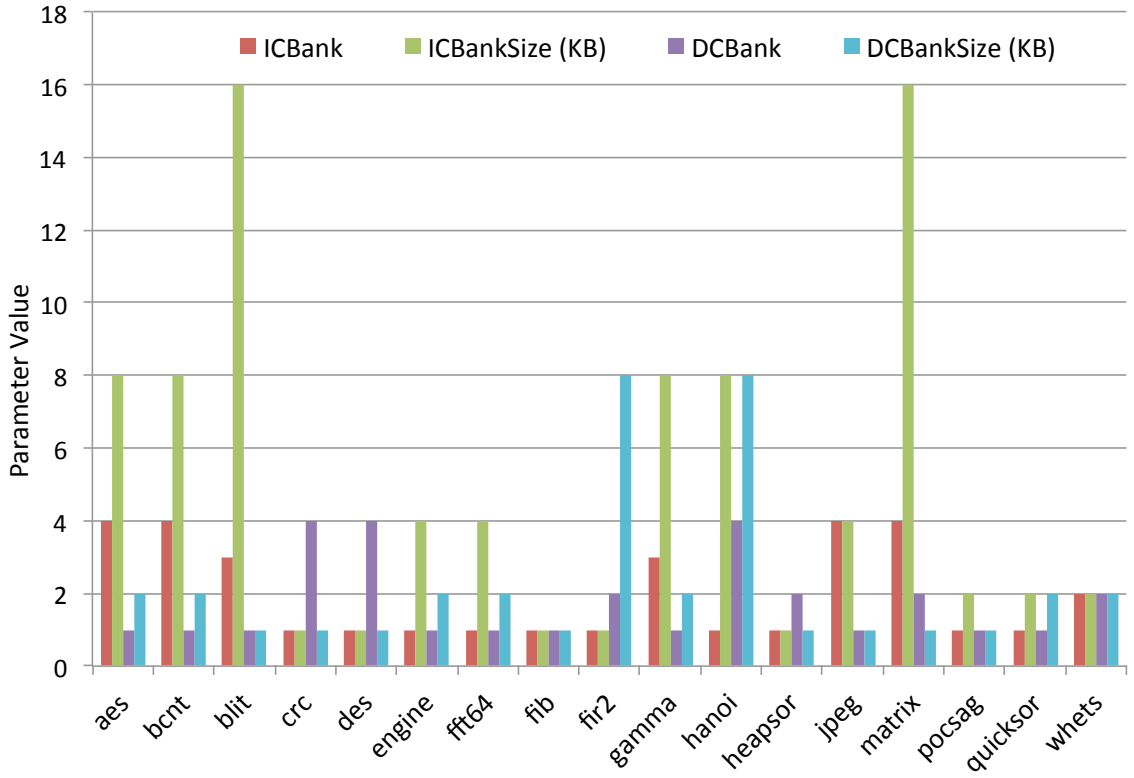


Figure 6.11: Single core: optimal parameters configuration with best application's performance

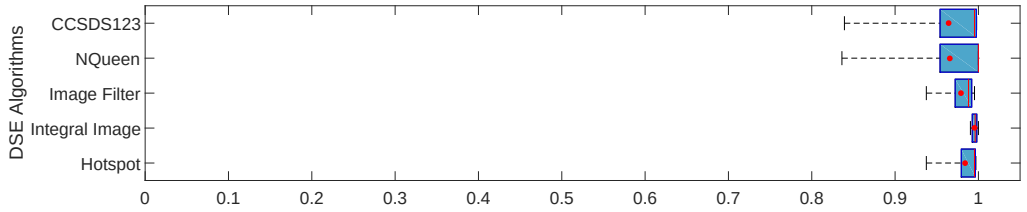
required by ES. Finally, the average number of simulations in "$Combined$" approach (by running all algorithms) is ≈38% of ES and the selection of desirable EDP is 99%. The results of simulated annealing showed a phenomenal performance within the DSE framework. However, these algorithms cannot take full benefit of parallel simulation of the presented framework whereas the FGS, a genetic-based algorithm, can exploit parallel execution of individual simulation runs. Its setup and behavior towards the presented architecture and use-cases is documented in the following section.

6.4 DSE Framework Performance Analysis of the Genetic-based FGS Optimization

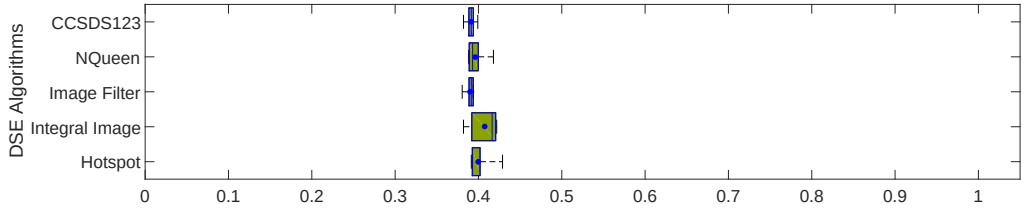
Similarly, to simulated annealing DSE optimizations, the FGS results are compared with those given by exhaustive search, where the comparison is based on two perspectives: DSE process duration and the accuracy of the given optimal solution. The DSE process is effectuated with the simulation of five multi-core applications selected from the use cases

FSG Parameter	Value
Population Size	50
Tournament Size	7
Mutation Threshold	0.5
Max Simulations	500
Max Generations	50

Table 6.2: FGS Framework Default Parameters



(a) Selection Accuracy (Normalized to ES)



(b) Number of Simulations (Normalized to ES)

Figure 6.12: Comparison between FGS and ES for limited simulation termination condition

described in Section 6.1; CCSDS123, Hotspot benchmark, Integral Image, Image Filter (moving average filter) and Nqueen puzzle problem. The FGS framework runs every multi-core application on the $leon3mp$ system with parameters configurations given in the Table 6.1. The termination condition for FGS algorithm is based on two factors: fixed simulation number and the given time-power budget. Each termination condition of FGS execute the test application five times and the mean values are presented subsequently.

6.4.1 Termination with a Limited Number of Simulations

In this approach, the designer specifies the maximum number of simulations that should not be exceeded in order to get the optimal design. In this work, 500 simulations were selected as the upper limit, so when the counter reaches 500 simulations, the framework finishes the execution of the current generation and reports the optimal solution. This

approach can be used to get an initial performance estimates for a bigger design space which helps to set performance-based termination condition for FGS framework.

Table 6.2 shows the default set of parameters used in the FGS framework, current implementation allows the duplication policy means one individual can be selected for multiple time. Since the evaluation is applied to all 50 individuals (which is the population size) in one generation, the maximum number of simulations could be reached at any step of that generation. Thus, the total number of simulation will be between $< MaxSimulations, MaxSimulations + PopulationSize - 1 >$.

Comparison between the FGS and by the ES is illustrated in Figure 6.12[3]. The distinction between the two approaches is very clear in terms of the optimal selection accuracy and number of simulations. The optimal parameters are selected ≈95% (at least) of accuracy and number of simulations performed are ≈44% (at most) compare to the ES.

6.4.2 Termination with a Fixed Time-power Budget

In this method, there is no limitation on the number of simulations. However, the designer fixes an execution time and power dissipation of the system under test based on the design objectives. When an individual among the best 10 individuals of the actual generation is trapped in this time-power interval, then the algorithm stops and gives this individual as the desired optimal solution. Time-power budget used in this work is the absolute minimum EDP obtained through the exhaustive search. However, optionally the estimate can be taken from a trial run of the FGS with a limited number of simulations for a given application.

The comparison between the simulation time of the FGS and the ES is given in Figure 6.13, the reduction in the number of simulation ranges between 95% (Integral Filter) and 67% (NQueen). Table 6.3 shows the speed of FGS convergence in finding desired EDP value. It is important to mention that this type of termination condition is relative because it requires a previous knowledge about the target application. And this is what makes the results in term of simulations better than the results given by the first termination condition method.

[3]Performance of DSE Algorithms: For selection accuracy higher is better and number of simulations lower is better)

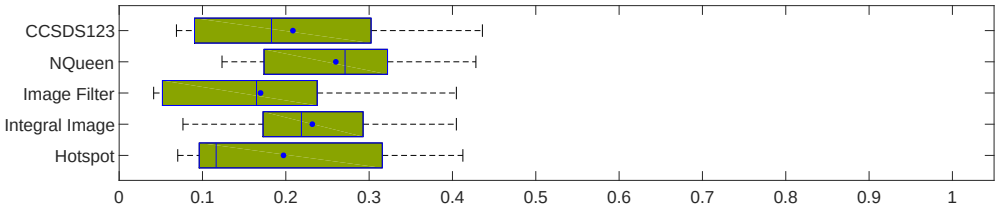


Figure 6.13: FGS Number of Simulations (Normalized to ES) for the time-power budget termination condition

Application	Number of generations	Number of simulations
CCSDS123	9	146
Hotspot	50	534
Intrgral image	20	271
Image filtering	12	166
NQueen	17	250

Table 6.3: 5 Tests average of FGS generations and simulations with termination condition time-power budget

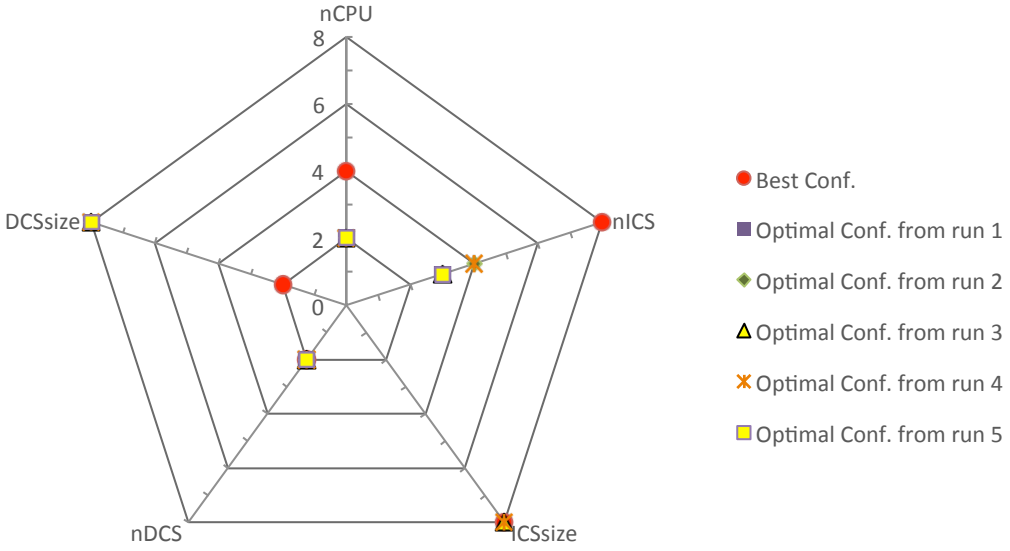


Figure 6.14: Optimal parameter accuracy of CCSDS123 through FGS with the 500 simulations limit

6.4.3 Optimal Parameter Accuracy

In addition to the number of simulation and the desired EDP selection, the accuracy of selected configuration parameters has to be investigated. Meaning, how much the given optimal solution differs to the absolute solution given by ES. In this work, the focus on the termination condition with a limited number of simulations, whereas the parameters configuration depends upon the designer selection of the desired EDP.

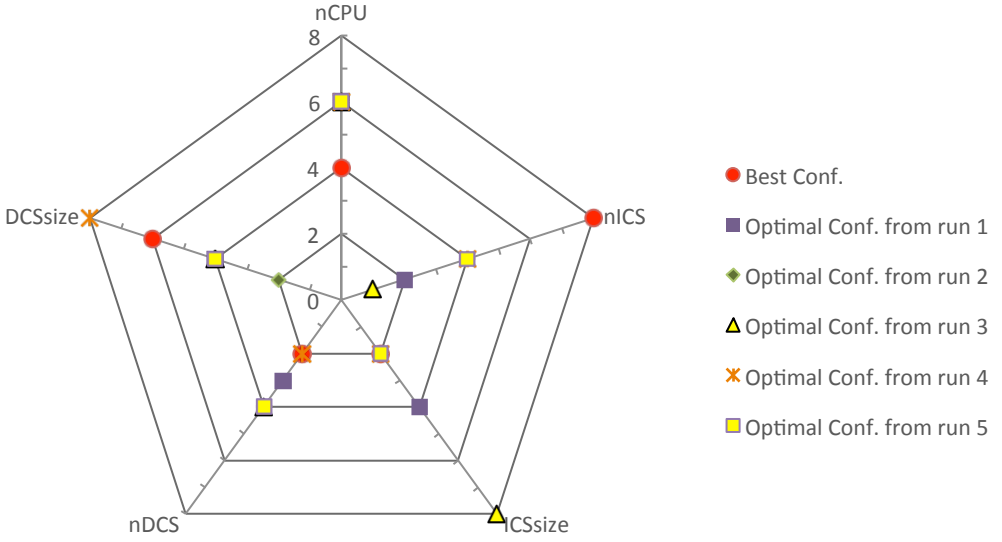


Figure 6.15: Optimal parameter accuracy of Hotpot benchmark through FGS with the 500 simulations limit

As mentioned before, five tests are run for every application which produces five optimal parameters configurations, as shown in the Figures 6.14 and 6.15 the five optimal solutions and the absolute best from ES are mapped on five-axis. Each axis represents an absolute best architectural parameter along with its counterparts from the five tests. The parameter value within the inner circle of the graphs represents less SoC resources than their outer circle, e.g. a cache size of $2KB$ is better in terms area and power than $4KB$.

Taking the example of the CCSDS123, in Figure 6.14, it can be seen that on four axes: p1(nCPU), p2(nICS), p3(ICSsize) and p4(nDCS) present either fully identified as the best parameter or more resources efficient (closer to center) than the optimal parameter. There is one miss match in the axis of p5(DCSsize), the obtained optimal architecture value is garter (away from center) than the absolute value. Figure 6.15 shows hotpot benchmark with one good match of p2(nICS) whereas rest of parameters have partial matches. These representations give the designer a valuable perspective of the system under test to look for the possibility to re-design the application with different architecture. The remaining applications are given in Appendix A.

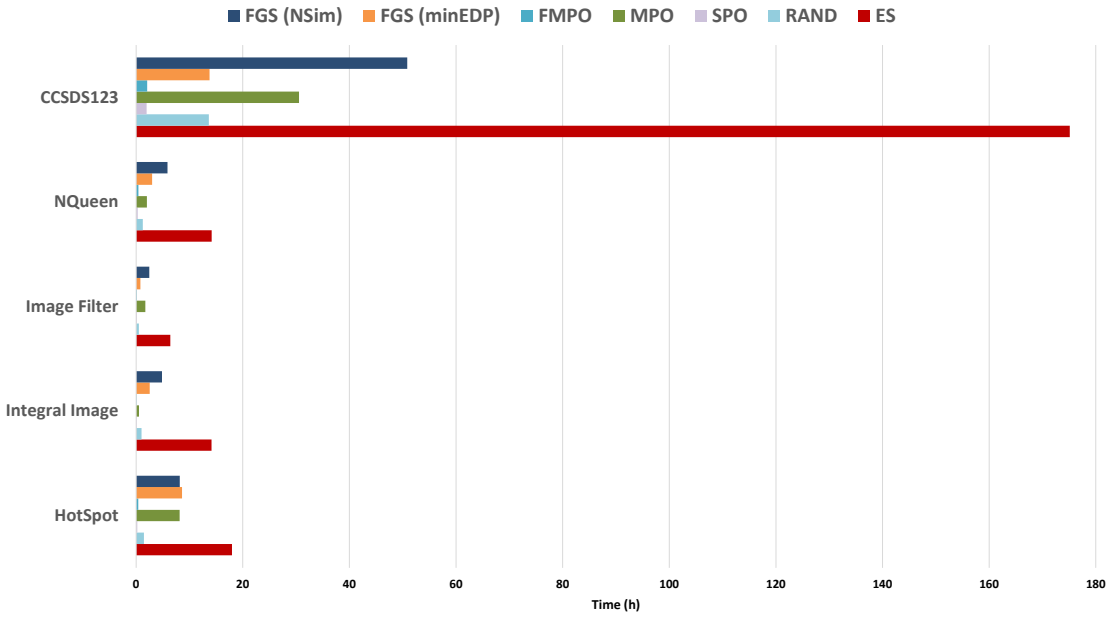
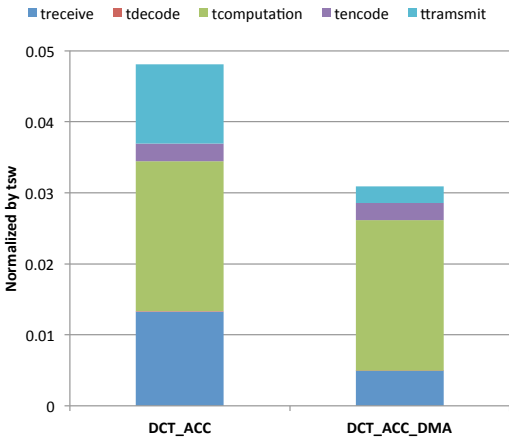


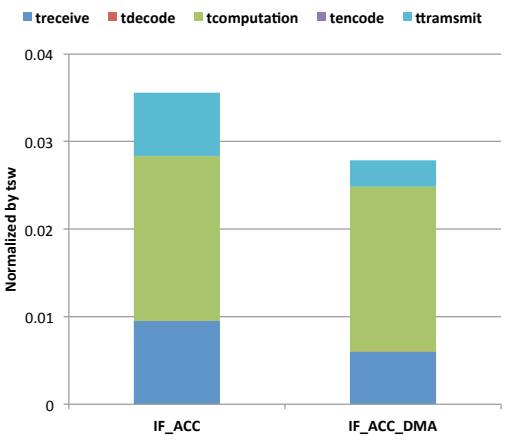
Figure 6.16: Total Simulation Time for finding an Optimal Design

6.5 Simulation Time Reduction Comparison

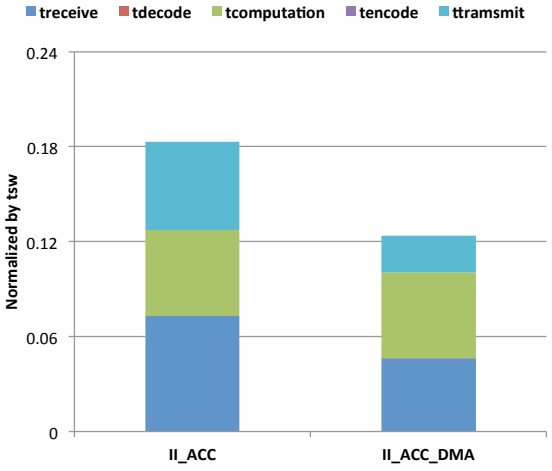
As shown in previous sections, the DSE framework considerably decreases the number of simulations for optimal design. This leads to a significant reduction in total design exploration time and the simulation logging output data. Illustration of the total DSE time reduction is shown in Figure 6.16, where x-axis shows the total hours spent in finding an optimal solution by the DSE algorithms. In the case of CCSDS123 application, the reduction in the total simulation time of 175 hours was reduced to 51 hours with FGS (NSim: limited simulations); 14 hours FGS (minEDP: limited time-power budget); 14 hour with RND; 2 hours with SPO; 31 hours with MPO and 3 hours with FPMO. The optimal solution is strongly based on the nature and requirements of the target application implementation. The results show the simulation time for optimal configuration is reduced to under 3% by FMPO whereas FGS algorithm better performance for a bigger design. The presented DSE framework executed the exploration process fully automated, which significantly reduces the overall design time. Although the design space exploration of hardware parameters gives an optimal design, it often does not meet the performance requirements of real-time. Therefore, the designer needs to analyze the possible hardware and software co-design.



(a) DCT accelerators



(b) Image filter accelerators

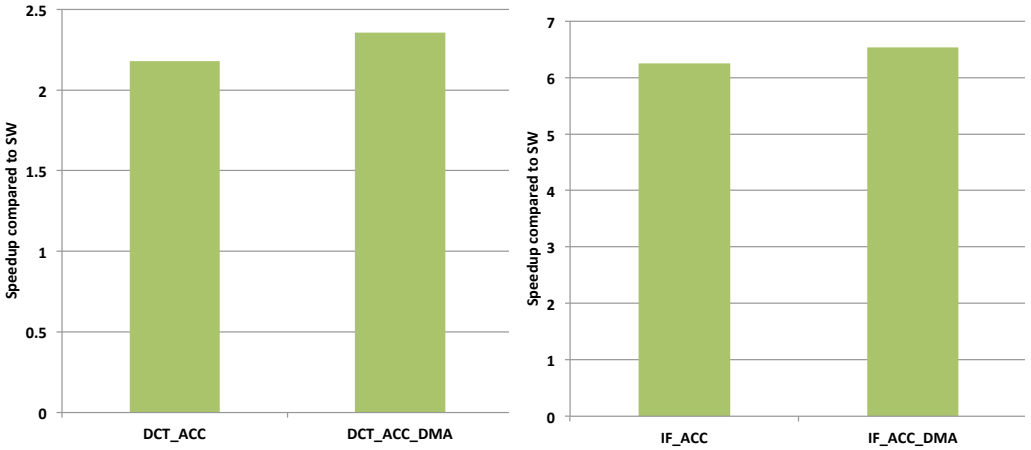


(c) Integral image accelerators

Figure 6.17: Combined accelerator delays

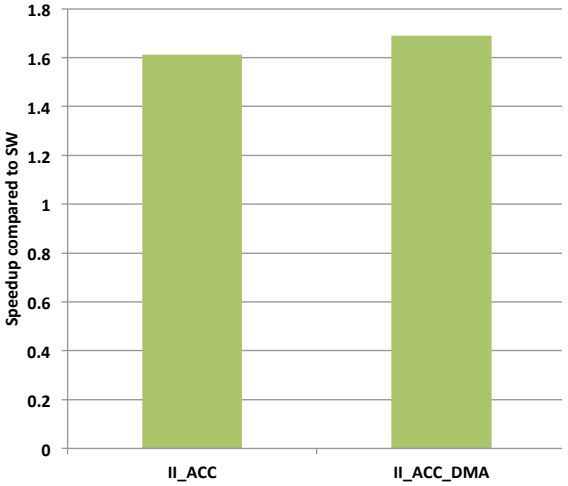
6.6 SW2TLM Performance Analysis

To demonstrate the hardware functionality of the SW2TLM automation framework, it is applied to three different applications: JPEG Codec, image filtering and image integration selected from the uses described in Section 6.1. For each, the performance of the accelerator with and without DMA was compared to the performance of a pure software solution. Figure 6.17 illustrates the difference in the ratio of communication overhead vs. actual computational delay. For a comprehensive analysis of the accelerator latency, the total computation time is broken down into the individual components shown in the previous chapter's table 5.2. The main findings are summarized as:



(a) DCT accelerators

(b) Image filter accelerators



(c) Integral image accelerators

Figure 6.18: Absolute application speed up

- **Decode/Encode:** It is directly visible, that "t_decode" and "t_encode" contribute an insignificant delay to the overall accelerator delay. This effect will be more prominent for the complex data types; the structure of the tested image formats is mostly sequential in nature and requires very little encoding and decoding effort. Of the tested application, the only visible encoding effort is observed for the DCT, here the data type conversion is necessary to account for the difference in endianness between hardware accelerator and the emulated LEON3 core.

- **Execution time (accelerated section only):** the results show the speedup of 5.46x with accelerator and 8.08x including the DMA for image integration, which is less

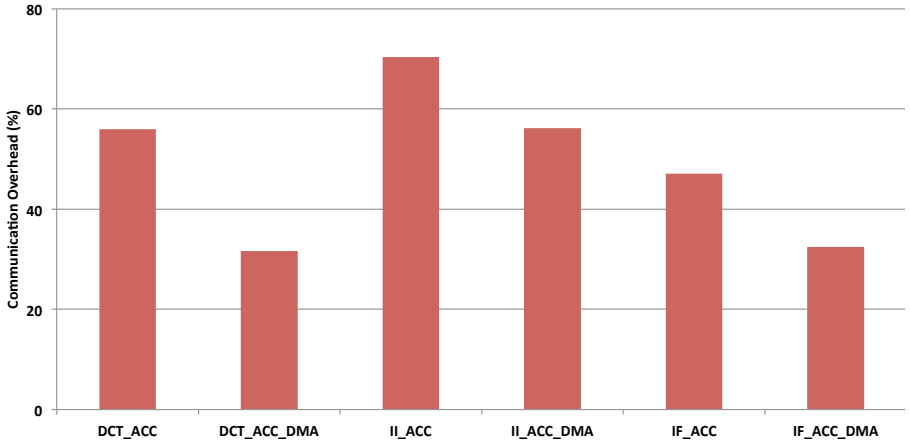


Figure 6.19: Communication overhead shown in percentage of complete accelerator delay

than the other two applications. The main reason for this difference is the underlaying test application is relativity simple, it requires mainly pixel summation operations. Whereas the DCT and image filtering contains complex operations such as convolution, which makes them better candidates for hardware acceleration. The DCT is 20.8x and image filtering is 28x faster using the accelerator (with DMA controller 32.3x and 35.9x) than the software implementation.

- **Absolute execution time:** Figure 6.18 shows the speedup factors achieved over the complete application execution time in software. The highest speedup up to 6.5x, was observed for the image filtering. The difference between software and DMA data handling accounts for up to 0.3x absolute speedup.

$$Co_t = \frac{t_{receive} + t_{decode} + t_{encode} + t_{transmit}}{t_{acc\_total}} \tag{6.4}$$

- **Communication overhead:** by using the total delay of the accelerator form equation 5.1, the overall communication of accelerators is calculated in equation 6.4. Figure 6.19 illustrates the communication overhead ratio recorded during data transfer between the CPU and accelerator in the simulation. As expected, the highest communication overhead was measured for the software-based data handling

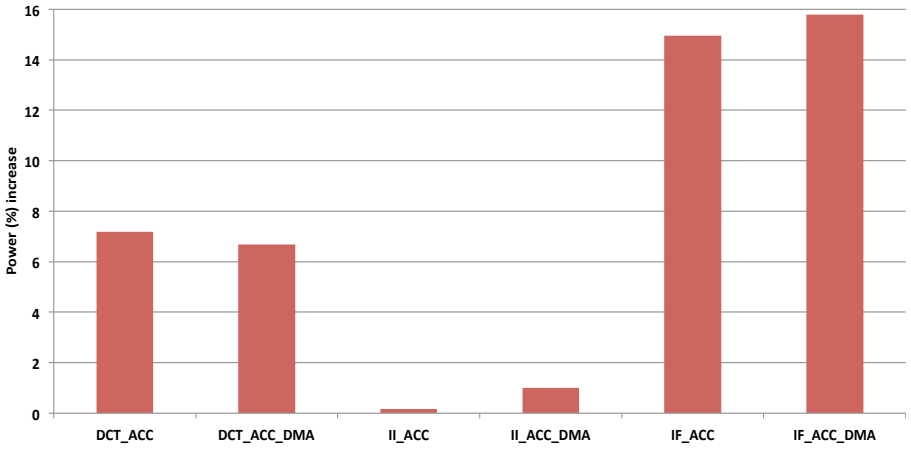


Figure 6.20: Power consumption increase to SW

(ranging from 47% up to 70%). When data movement is handled by the DMA controller a reduction of at least 14% was observed.

- **Power and Energy:** The power model applied to the ISS does not incorporate typical power saving techniques when idle. Modeling these techniques on the system-level is part of the outlook of this research. Hence, the average power draw increases as shown in Figure 6.20, when the accelerator hardware is added to the system. This increase ranges from 0.2% up to 19%. The high values of the image filtering are caused by frequent data accesses related to the size of the filter kernel. Despite this limitation, the overall energy consumption (see Figure 6.21) exhibits a significant improvement. The observed energy consumption ranges between 18% and 62% of the original software implementation.

6.7 Summary

This chapter presented a use cases implementation from real-image resource-hungry applications. Which are applied to a multi-core architecture based on the $leon3mp$ CPU core. The fully automated DSE framework explored the design space of the multi-core system with a range of configuration parameters that include the number of cores, data

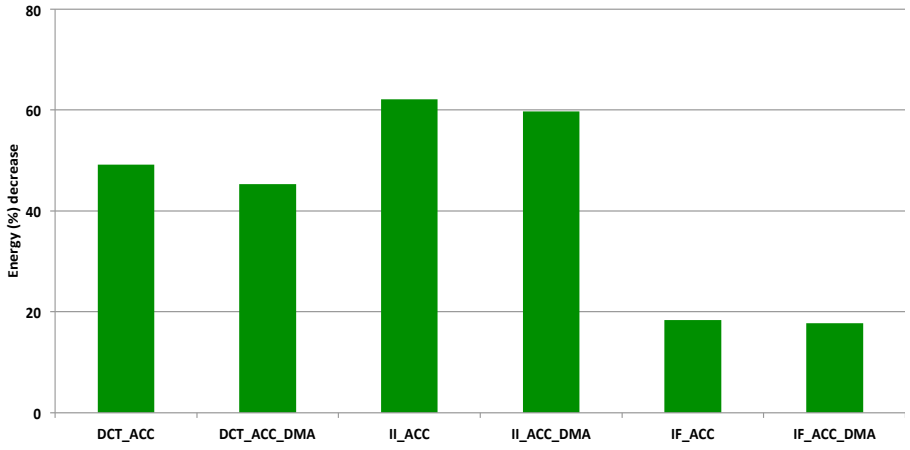


Figure 6.21: Energy consumption relative to SW

and instruction caches using the SocRocket virtual platform. The design space was defined over a set of functional parameters and evaluated based on its non-functional characteristics. Here, different DSE algorithms were compared, a comprehensive analysis was carried out in terms of decrease in simulations number, selection accuracy of desired EDP and the resulting quality compared to the best configuration identified by an exhaustive search. Lastly, benefits of the SW2TLM framework for the automation of hardware/software-partitioning was demonstrated. Focusing on execution speed-up, communication overhead, and power dissipation of the target SoC architecture.

Chapter 7

Conclusion and Future Work

Three interlocked tools were developed and evaluated in the presented work: beginning with a versatile power estimation framework, sophisticated optimization techniques were applied to efficiently explore the design space surmounting into an automatic hardware/-software partitioning and exploration tool. Each of which is summarized below.

The system-level designer needs to explore high numbers of architectural configurations to identify an optimal or close to optimal design. For this task, RTL power estimates derived from synthesis reports are tailored to one specific architectural configuration and therefore cannot be used directly on the system-level. The power estimation models presented in this work are therefore based on normalized values, calculated using post place and route synthesis information generated using generic RTL configurations. Allowing re-use in different designs without the need for additional RTL synthesis iterations.

The key to accurate power estimations is the observance of two major power dissipation types: the static power representing the leakage of the cell and heavily dependent on technology; the dynamic power, which is composed of internal (scale with the frequency and chip area) and switching power (application depended). To this end, a Framework was designed to integrate the power estimation models into a fast simulation environment. It provides features such as instantaneous power reporting which enables the designer to detect power draw extremes earlier and in high resolution compared to traditional design processes. To further improve the usability and productivity, it allows seamless creation and integration of new power models and update of existing ones when the technology

node is changed without complete re-compiling of the underlying SystemC virtual platform. Capturing power consumption data in high detail and fine granularity generates large amounts of data for system-level simulations. To avoid bottlenecks of slow text-based logging, a flexible HDF5 database interface was added into the Framework. The database accelerates the logging and analysis, simulation results can be visualized in an interactive GUI which supports direct comparison of individual component power consumption data as well as storing the selected results in the form PNG or CSV spreadsheets for later analysis.

Evaluation of the power estimation methodology was performed within the ASIP [14] design flow, for which a mean power estimation error was as low as 15% was achieved. This was done using a generic 90nm ASIC design kit, allowing even lower estimation errors when transitioning to a more specific technology library. The high accuracy of the estimation results enables designers to speed up the architectural exploration process by transitioning to the system-level, removing the need for time-consuming RTL synthesis and power simulations.

The system-level design driven by virtual platforms or prototypes enables fast simulation to explore larger numbers of architectural parameter permutations compared to traditional RTL simulations. However, manual exploration of large numbers of architectural parameters takes time (many hours if not days) and strongly relies on the experience of the designer. In this context, an automatic and fast design space exploration extension for virtual platforms was developed to reduce the number of necessary simulations to reach a suitable parameter configuration. Several traditional optimization techniques were investigated: simulated annealing-based heuristics that includes Random Search (RND), Single Parameter Optimization (SPO), Multi-Parameter Optimization (MPO) and Fast Multi-Parameter Optimization (FMPO). In addition, Fast Genetic Search (FGS) adopted from the field of evolutionary computing (genetic) was implemented as an optimization strategy for the presented extension. Evaluation of these optimization strategies was accomplished by using different applications mapped to a SoCRocket configuration with multiple LEON3 (1-6) processor systems running RTEMS OS as well as bare-metal benchmarks running on a single processor system. Different configurations (in total 1280) of five architectural parameters: number of processors, instruction cache banks, instruction cache bank size, data cache banks and data cache bank size were selected to

span the design space. The applications are evaluated for system power draw and application simulation time in order to emphasize requirements of real-time computation, the energy-delay product is used as the main performance metric for the DSE. The presented Framework applied the above-mentioned strategies to find the optimal parameter configurations with minimum designer intervention and lowest number of actually performed simulation runs.

A significant reduction in exploration time by ≈62% (worst case) was observed. In addition, the selection accuracy of optimal parameters reached roughly 95% (FSG, with user-defined stop condition) and ≈90% (combined all other remaining algorithms) compared to the absolute solution identified by exhaustive search in both single and multi-processor DSE.

The hardware architectural parameters exploration is not enough to meet the real-time performance constraints of complex applications, which require an efficient hardware software co-design. Exploring possible hardware/software scenarios is time consuming and tedious job. To address this challenge, a flexible and automated workflow "SW2TLM" for generating system level hardware accelerators from software applications and their integration to virtual platform framework is presented. It supports engineers by enabling efficient exploration of different partitioning scenarios with minimal effort, all complexity of the hardware architecture and software integration is hidden from the user. Users are only required to characterize the investigated software section to generate a corresponding hardware accelerator model embedded into a complete virtual prototype on the TL-level. The framework leverages a virtual platform to explore the accelerator's behavior and the overhead incurred from the communication. The workflow is demonstrated using the SoCRocket virtual platform executing different image processing applications. The framework reported communication, performance and power parameters are a valuable foundation to base future architecture decisions on. The presented evaluation illustrated how important it is to include communication overhead into the exploration space, as it can account for most of the overall accelerator computation time (here up to 70%). Also, capturing accurate overall energy consumption is essential for the designing constraints or battery dependent domains.

7.1 Future Work

The presented work demonstrates a set of solutions for current challenges in SoC design. However, the continuously growing complexity and physical limitations require further research and exploration into fast and accurate power estimation models. The research area is moving fast and opens many new opportunities to improve the design and development processes. Of those, the following section will illuminate some of the high importance and strong relation to the presented material.

Cost functions are a central requirement for accurate design space exploration, adding more and more detail to it will improve the exploration process as well as the resulting architecture. Combining detailed chip area information with precise power and thermal models allows earlier detection of local hot-spots and peak currents to better anticipate the Dark-Silicon effects in the final product. Here the presented framework provides high flexibility and will allow easy integration of new models independent of the parameter characteristics. In addition, for highly complex and strongly heterogeneous architectures of the future, further research into new optimization algorithms is required. Strategies as deep learning can be applied to tackle design spaces significantly larger than what is possible today.

Beyond viewing the power models in isolation, extending the underlying virtual platform or porting the presented work to a simulation environment supporting modern low-power techniques will further improve the estimation accuracy. These techniques are tightly interwoven with architectural features of modern multi-processor SoCs and present new challenges for designers of virtual platforms. Accurate simulation of dynamic frequency scaling, voltage scaling, smart deactivation of unused on-chip resources will require close interdisciplinary cooperation between IP designers and power modeling experts.

Furthermore, tracing of software to identify software sections for acceleration can improve hardware/software solutions. In this context, the communication between software and hardware resources is of high importance as the movement of complex data structures might cancel out the performance gains of customized hardware. This issue is known in the HLS domain, as data handling and organization in software observes a far greater degree of freedom compared to hardware implementation.

List of Figures

List of Tables

Appendix A

Results

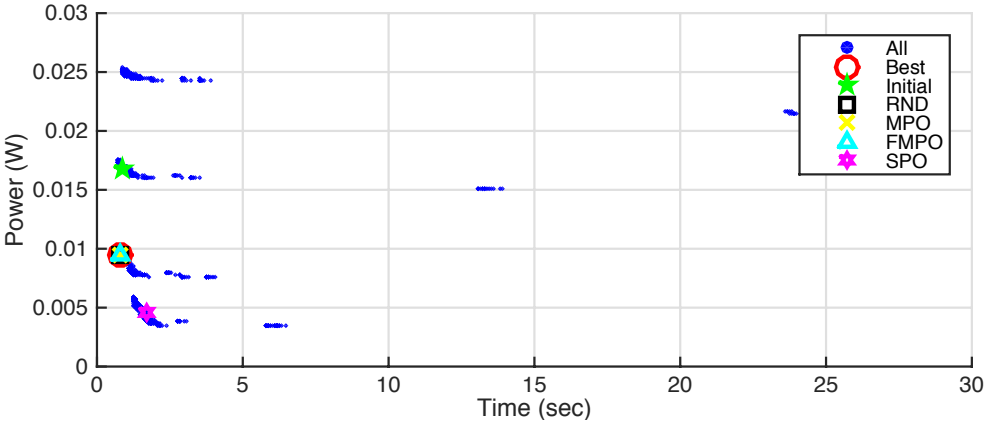


Figure A.1: Multi-core System DSE: CCSDS123

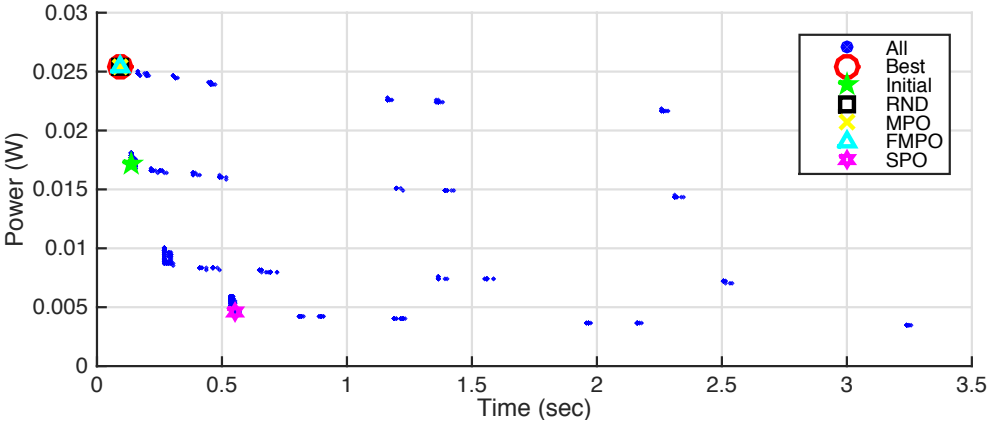


Figure A.2: Multi-core System DSE: HotSpot

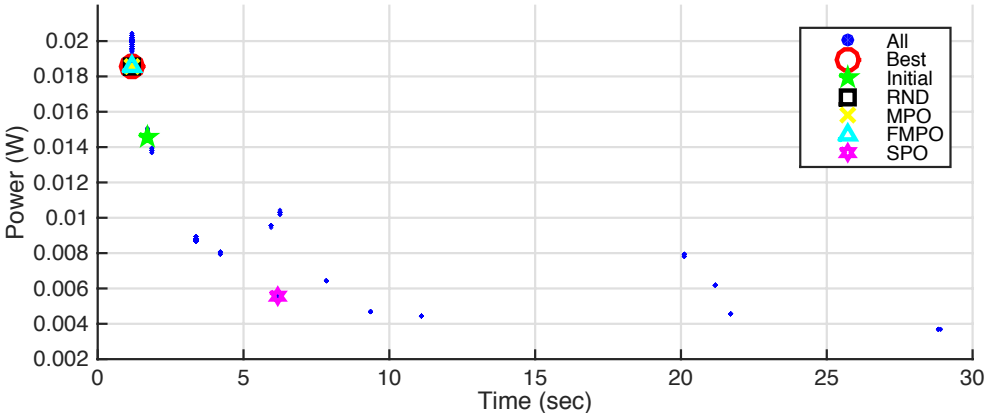


Figure A.3: Multi-core System DSE: NQueen

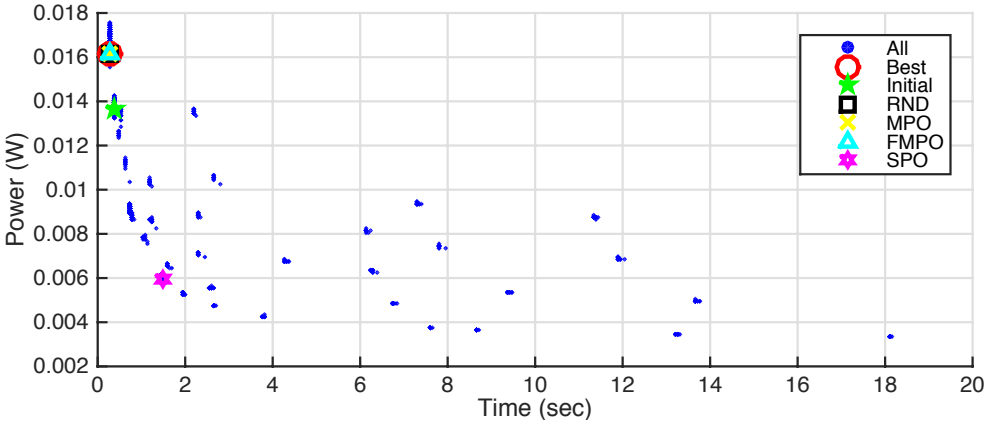


Figure A.4: Multi-core System DSE: Image Filter

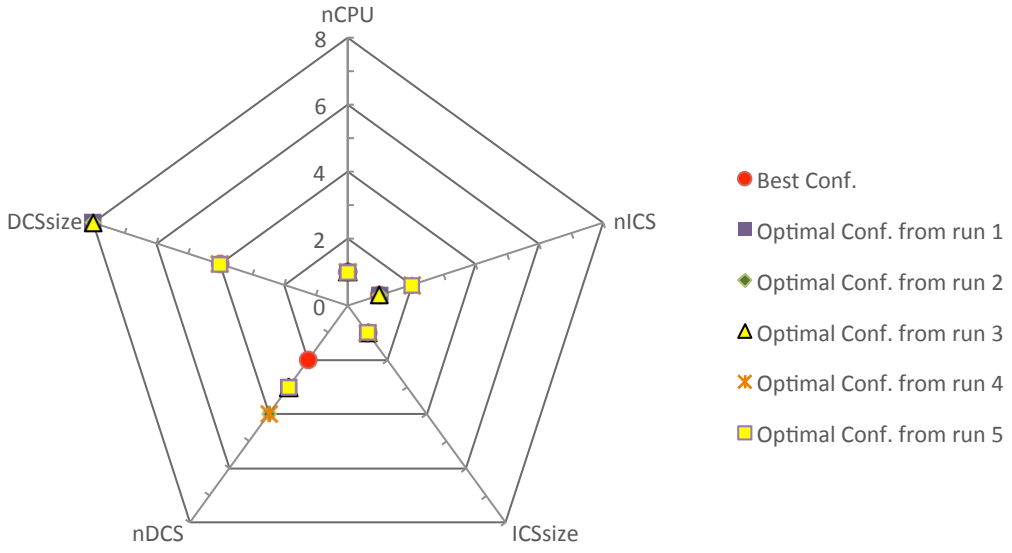


Figure A.5: Optimal parameter accuracy of Integral Image through FGS with the 500 simulations limit

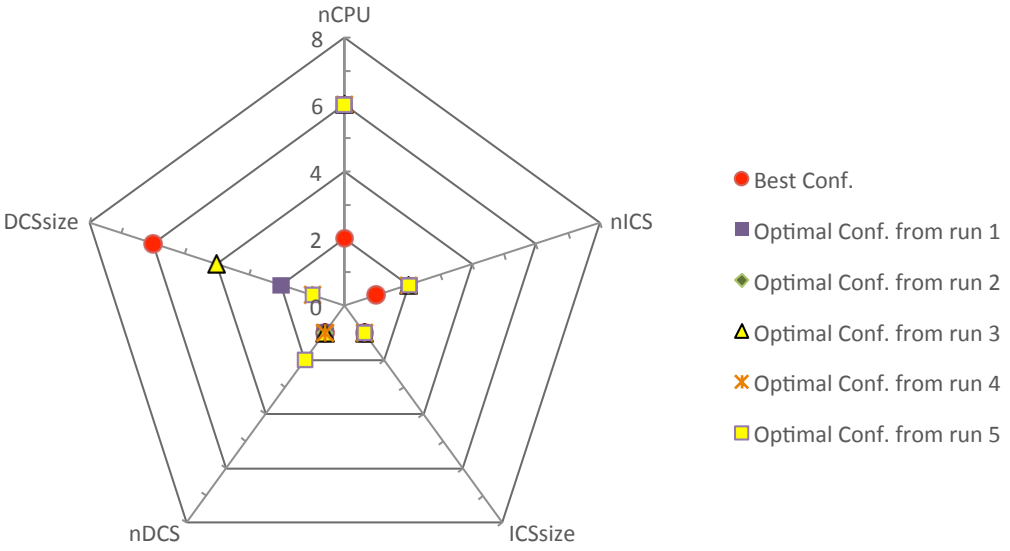


Figure A.6: Optimal parameter accuracy of NQueen through FGS with the 500 simulations limit

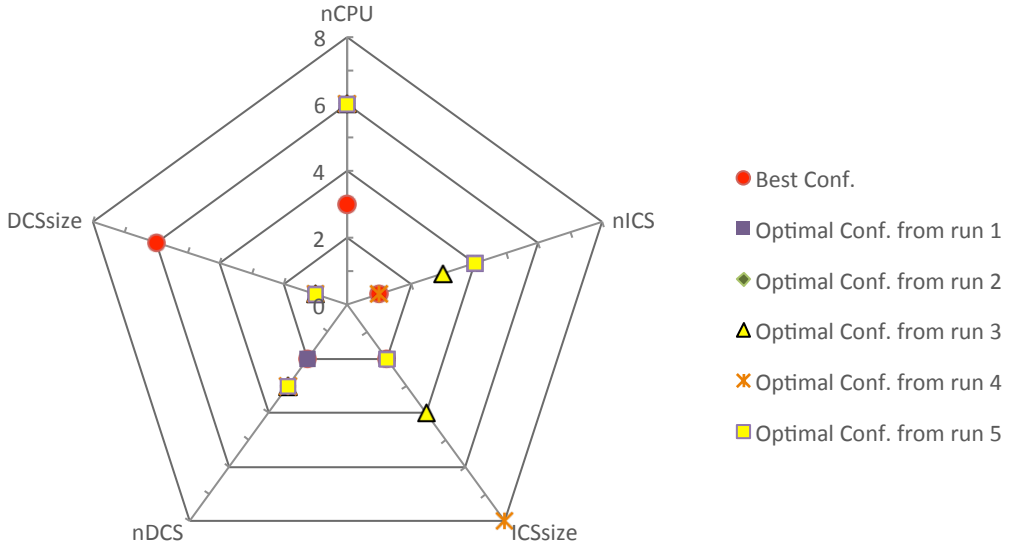


Figure A.7: Optimal parameter accuracy of Image Filter through FGS with the 500 simulations limit

Bibliography

[1] Zebra. Reinventing retail:2017 retail vision study. 2017. URL https://www.zebra.com.

[2] Ron Collett and Dorian Pyle. What happens when chip-design complexity outpaces development productivity? *McKinsey on Semiconductors*, 3:24–33, 2013.

[3] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[4] Robert H. Dennard, Fritz H. Gaensslen, Hwa nien Yu, V. Leo Rideout, Ernest Bassous, Andre, and R. Leblanc. Design of ion-implanted mosfets with very small physical dimensions. *IEEE J. Solid-State Circuits*, page 256, 1974.

[5] Intel. Tick-Tock Model. http://www.intel.com/content/www/us/en/silicon-innovations/intel-tick-tock-model-general.html, 2017.

[6] Intel Technology and Manufacturing. https://newsroom.intel.com/press-kits/leading-edge-intel-technology-manufacturing/?_ga=1.119286601.172518219.1491910951, 2017.

[7] Christian Märtin. Post-dennard scaling and the final years of moore's law consequences for the evolution of multicore-architectures. *Informatik und Interaktive Systeme*, 2014.

[8] Original data up to the year 2010 collected, O. Shacham-K. Olukotun L. Hammond plotted by M. Horowitz, F. Labonte, C. Batten New plot, and data collected for 2010-2015 by K. Rupp. 40 years of microprocessor trend data. https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/, 2013.

[9] ITRS2.0. International technology roadmap for semiconductors 2.0. 2015.

[10] anandtech. Apple A10 (IPhone 7) . http://www.anandtech.com/show/10658/apple-announces-iphone-7-iphone-7-plus, 2017.

[11] G. Martin. Overview of the mpsoc design challenge. In *2006 43rd ACM/IEEE Design Automation Conference*, 2006. doi: 10.1109/DAC.2006.229245.

[12] Open SystemC Initiative (OSCI) TLM-2.0 standard. OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL. 2008.

[13] T. Schuster, R. Meyer, R. Buchty, L. Fossati, and M. Berekovic. SoCRocket – A virtual platform for the European Space Agency's SoC development. In *ReCoSoC, 9th International Symposium on*, 2014. doi: 10.1109/ReCoSoC.2014.6860690.

[14] S. A. A. Shah, J. Wagner, T. Schuster, and M. Berekovic. A lightweight-system-level power and area estimation methodology for application specific instruction set processors. In *2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–5, Sept 2014.

[15] S. A. A. Shah, B. Farkas, R. Meyer, and M. Berekovic. Accelerating mpsoc design space exploration within system-level frameworks. In *2016 IEEE Nordic Circuits and Systems Conference (NORCAS)*, pages 1–6, Nov 2016.

[16] Syed Abbas Ali Shah, Sven Horsinka, Bastian Farkas, Rolf Mayer, and Mladen Berekovic. Automatic exploration of hardware/software partitioning. In *Design and Verification Conference (DVCon) United States 2017*, 2017.

[17] Alexander Silbey Aaron Aboagye, Dorian Pyle. By the numbers: R&d productivity in the semiconductor industry. *McKinsey on Semiconductors*, pages 49–54, 2014.

[18] B. Bailey and G. Martin. *TESL Models and Their Application: Electronic System Level Design and Verification in Practice (Embedded Systems)*. Springer US, 2009.

[19] Frank Ghenassia. *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer US, 2005.

[20] SYNOPSYS Platform Architect MCO. https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html, 2017.

[21] Cadence Virtual System Platform (VSP). https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/software-driven-verification/virtual-system-platform.html.

[22] Modelling ARM-Based SoCs and Subsystems with Fast Models. https://developer.arm.com/products/system-design/fast-models, 2017.

[23] Imperas simulators. http://www.imperas.com/, 2017.

[24] Mentor VISTA Virtual Prototyping. https://www.mentor.com/esl/vista/virtual-prototyping/, 2017.

[25] M. D. Grammatikakis, A. Papagrigoriou, P. Petrakis, and G. Kornaros. Monitoring-aware virtual platform prototype of heterogeneous noc-based multicore socs. In *2013 Euromicro Conference on Digital System Design*, pages 497–504, Sept 2013.

[26] Young-Ran Lee, Sang-Young Cho, and Jeong-Bae Lee. The design of a virtual prototyping based on armulator. In *Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science*, pages 387–390, Washington, DC, USA, 2005.

[27] The ARMulator. http://infocenter.arm.com/help/topic/com.arm.doc.dai0032f, 2018.

[28] M.T. Yourst. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. *2007 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 23–34, 2007.

[29] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh

Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. pages 1–7, 2011.

[30] Sven Alexander Horsinka, Rolf Meyer, Jan Wagner, Rainer Buchty, and Mladen Berekovic. On rtl to tlm abstraction to benefit simulation performance and modeling productivity in noc design exploration. pages 39–44, 2014.

[31] Ieee standard system c language reference manual. *IEEE Std 1666-2005*, pages 01–423, 2006. doi: 10.1109/IEEESTD.2006.99475.

[32] L. Cai and D. Gajski. Transaction level modeling: an overview. In *First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis (IEEE Cat. No.03TH8721)*, pages 19–24, 2003.

[33] Mark Burton, James Aldis, Robert Günzel, and Wolfgang Klingauf. Transaction level modelling: A reflection on what tlm is and how tlms may be classified, 2007.

[34] Adam Rose, Stuart Swan, John Pierce, and Jean michel Fern. Transaction level modeling in systemc. Technical report, Open SystemC Initiative, 2005.

[35] Adam Donlin. Transaction level modeling: Flows and use models. In *Proceedings of the 2Nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 75–80. ACM, 2004.

[36] SoCRocket: Transaction-Level Modeling Framework for Space Applications. https://github.com/socrocket, 2018.

[37] GreenSocs Open Source Technology. https://www.greensocs.com, 2018.

[38] GreenReg. http://docs.greensocs.com/greenlib/greenreg/introduction/introduction.html, 2018.

[39] S. Swan and J. Cornet. Beyond tlm 2.0: New virtual platform standards proposals from st and cadence. *NASCUG at DAC*, 2012.

[40] Luca Fossati. TLM2.0 Standard into Action: Designing Efficient Processor Simulators. In *Proc. of 19th IP based electronic conference and exhibition (IP-SoC)*, 2010.

[41] R. Meyer, J. Wagner, R. Buchty, and M. Berekovic. Universal scripting interface for systemc. In *DVCon Europe Conference Proceedings 2015*, 2015.

[42] Aeroflex Gaisler GRLIB IP Library. http://www.gaisler.com/index.php/downloads/leongrlib, 2018.

[43] J. Wagner, R. Meyer, R. Buchty, and M. Berekovic. A scriptable, standards-compliant reporting and logging extension for systemc. In *SAMOS, 2015 International Conference on*, 2015. doi: 10.1109/SAMOS.2015.7363700.

[44] D. Lidsky and J. M. Rabaey. Early power exploration-a world wide web application [high-level design]. In *33rd Design Automation Conference Proceedings, 1996*, 1996.

[45] R. A. Bergamaschi and Y. W. Jiang. State-based power analysis for systems-on-chip. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, 2003.

[46] B. Fischer, C. Cech, and H. Muhr. Power modeling and analysis in early design phases. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014.

[47] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference (DAC)*, pages 340–345, 2000.

[48] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pages 83–94, 2000.

[49] Teemu Pitkänen, Tommi Rantanen, Andrea Cilio, and Jarmo Takala. Hardware cost estimation for application-specific processor design. pages 212–221, 2005.

[50] Henk Corporaal and Marnix Arnold. Using transport triggered architectures for embedded processor design. pages 19–38, 1998.

[51] Luca Negri and Andrea Chiarini. Power simulation of communication protocols with statec. In *Applications of Specification and Design Languages for SoCs*, pages 277–294, 2006.

[52] Sumit Ahuja, Deepak Mathaikutty, Sandeep K. Shukla, and Ajit Dingankar. Assertion-based modal power estimation. pages 3–7, 2007.

[53] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. Regression-based rtl power modeling. *ACM Trans. Design Autom. Electr. Syst.*, pages 337–372, 2000.

[54] M. F. da S. Oliveira, L. B. de Brisolara, L. Carro, and F. R. Wagner. Early embedded software design space exploration using uml-based estimation. In *Seventeenth IEEE International Workshop on Rapid System Prototyping (RSP'06)*, 2006.

[55] Marcio F. S. Oliveira, Eduardo W. Brião, Francisco A. Nascimento, and Flávio R. Wagner. Model driven engineering for mpsoc design space exploration. In *Proceedings of the 20th Annual Conference on Integrated Circuits and Systems Design*, 2007.

[56] Tero Arpinen, Erno Salminen, Timo D. Hämäläinen, and Marko Hännikäinen. Marte profile extension for modeling dynamic power management of embedded systems. *J. Syst. Archit.*, 2012.

[57] C. Gomez, J. DeAntoni, and F. Mallet. Power consumption analysis using multi-view modeling. In *2013 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2013.

[58] Feriel Ben Abdallah, Chiraz Trabelsi, Rabie Ben Atitallah, and Mourad Abed. Model-driven approach for early power-aware design space exploration of embedded systems. *J. Signal Process. Syst.*, 2017.

[59] G. Onnebrink, R. Leupers, G. Ascheid, and S. Schürmans. Black box esl power estimation for loosely-timed tlm models. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 366–371, 2016.

[60] K. Grüttner, P. A. Hartmann, T. Fandrey, K. Hylla, D. Lorenz, S. Stattelmann, B. Sander, O. Bringmann, W. Nebel, and W. Rosenstiel. An esl timing & power estimation and simulation framework for heterogeneous socs. pages 181–190, 2014.

[61] T. Bouhadiba, M. Moy, F. Maraninchi, J. Cornet, L. Maillet-Contoz, and I. Materic. Co-simulation of functional systemc tlm models with power/thermal solvers.

In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 2176–2181, 2013.

[62] Intel Docea solutions. https://www.intel.com/content/www/us/en/system-modeling-and-simulation/docea, 2018.

[63] Yang Liu, Chenchen Deng, Leibo Liu, Shouyi Yin, and Shaojun Wei. System level power modeling of reconfigurable processor using the least square method. In *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 1–3, 2014.

[64] Marco Caldari, Massimo Conti, Marcello Coppola, Paolo Crippa, Simone Orcioni, Lorenzo Pieralisi, and Claudio Turchetti. System-level power analysis methodology applied to the amba ahb bus. 2003.

[65] Nikhil Bansal, Kanishka Lahiri, Anand Raghunathan, and Srimat T. Chakradhar. Power monitors: A framework for system-level power estimation using heterogeneous power models. pages 579–585, 2005.

[66] Marcello Lajolo, Anand Raghunathan, Sujit Dey, and Luciano Lavagno. Efficient power co-estimation techniques for system-on-chip design. In *IN PROC. DESIGN AUTOMATION and TEST EUROPE (DATE) CONF*, pages 27–34, 2000.

[67] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 469–480, 2009.

[68] CACTI: An integrated cache, area leakage memory access time, cycle time, and dynamic power model. http://www.hpl.hp.com/research/cacti/, 2018.

[69] G. B. Vece, M. Conti, and S. Orcioni. Pk tool 2.0: a systemc environment for high level power estimation. In *2005 12th IEEE International Conference on Electronics, Circuits and Systems*, 2005.

[70] S. K. Rethinagiri, R. B. Atitallah, and J. L. Dekeyser. A system level power consumption estimation for mpsoc. In *2011 International Symposium on System on Chip (SoC)*, 2011.

[71] D. Greaves and M. Yasin. Tlm power3: Power estimation methodology for systemc tlm 2.0. In *Proceeding of the 2012 Forum on Specification and Design Languages*, 2012.

[72] Felipe Klein, Rodolfo Azevedo, Luiz Santos, and Guido Araujo. *SystemC-Based Power Evaluation with PowerSC*, pages 129–144. 2011.

[73] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multigranular simulation. In *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems*, 2002. ISBN 1-58113-527-0. doi: 10.1145/513829.513835.

[74] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri, and L. Benini. An integrated open framework for heterogeneous mpsoc design space exploration. In *DATE*, 2006. doi: 10.1109/DATE.2006.244000.

[75] Lech Jozwiak, Menno Lindwer, Rosilde Corvino, Paolo Meloni, Laura Micconi, Jan Madsen, Erkan Diken, Deepak Gangadharan, Roel Jordans, Sebastiano Pomata, Paul Pop, Giuseppe Tuveri, Luigi Raffo, and Giuseppe Notarangelo. Asam: Automatic architecture synthesis and application mapping. *Microprocessors and Microsystems*, 2013. ISSN 0141-9331. doi: 10.1016/j.micpro.2013.08.006.

[76] S. Stuijk, M. Geilen, B. Theelen, and T. Basten. Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, July 2011. doi: 10.1109/SAMOS.2011.6045491.

[77] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, pages 35:1–35:44, 2011. ISSN 0360-0300. doi: 10.1145/1978802.1978814.

[78] Arne Hamann, Marek Jersak, Kai Richter, and Rolf Ernst. A framework for modular analysis and exploration of heterogeneous embedded systems. pages 101–137, 2006.

[79] F. Herrera and I. Sander. Combining analytical and simulation-based design space exploration for time-critical systems. In *Specification Design Languages (FDL), 2013 Forum on*, 2013.

[80] Z. J. Jia, A. Núñez, T. Bautista, and A. D. Pimentel. A two-phase design space exploration strategy for system-level real-time application mapping onto mpsoc. *Microprocess. Microsyst.*, 2014. ISSN 0141-9331. doi: 10.1016/j.micpro.2013.10.005.

[81] Matteo Monchiero, Ramon Canal, and Antonio González. Design space exploration for multicore architectures: a power/performance/thermal view. In *Proceedings of the 20th Annual International Conference on Supercomputing*, Cairns, Queensland, Australia, 2006. ACM.

[82] C. Ykman-Couvreur, J. Lambrecht, D. Verkest, F. Catthoor, A. Nikologiannis, and G. Konstantoulakis. System-level performance optimization of the data queueing memory management in high-speed network processors. In *DAC, 2002. Proceedings. 39th*, 2002. doi: 10.1109/DAC.2002.1012680.

[83] A. Baghdadi, N. Zergainoh, W. Cesario, T. Roudier, and A. A. Jerraya. Design space exploration for hardware/software codesign of multiprocessor systems. In *RSP 2000. 11th International Workshop on*, 2000. doi: 10.1109/IWRSP.2000.854975.

[84] A. Mahapatra and B. C. Schafer. Machine-learning based simulated annealer method for high level synthesis design space exploration. In *Proceedings of the 2014 Electronic System Level Synthesis Conference (ESLsyn)*, 2014.

[85] CyberWorkBench: High Level Synthesis from C/C++/SystemC to ASIC/FPGA. https://www.nec.com/en/global/prod/cwb, 2018.

[86] B. C. Schafer, Takashi Takenaka, and Kazutoshi Wakabayashi. Adaptive simulated annealer for high level synthesis design space exploration. In *2009 International Symposium on VLSI Design, Automation and Test*, 2009.

[87] N. V. Karanjkar and M. P. Desai. An approach to discrete parameter design space exploration of multi-core systems using a novel simulation based interpolation

technique. In *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2015.

[88] J. W. d. Mesquita, M. O. d. Cruz, M. M. Pereira, and M. E. Kreutz. Design space exploration using utnocs and genetic algorithm. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2016.

[89] S. Graf, M. Glaß, J. Teich, and C. Lauer. Multi-variant-based design space exploration for automotive embedded systems. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014.

[90] B. L. Mediouni, S. Niar, R. Benmansour, K. Benatchba, and M. Koudil. A bi-objective heuristic for heterogeneous mpsoc design space exploration. In *2015 10th International Design Test Symposium (IDT)*, 2015.

[91] M. K. Papamichael, P. Milder, and J. C. Hoe. Nautilus: Fast automated ip design space search using guided genetic algorithms. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[92] Hsiung Pao-Ann, Lin Chao-Sheng, and Liao Chih-Feng. Perfecto: A systemc-based design-space exploration framework for dynamically reconfigurable architectures. *ACM Trans. Reconfigurable Technol. Syst.*, 2008.

[93] M Santambrogio. *Hardware-Software codesign methodologies for dynamically reconfigurable systems*. PhD thesis, PhD thesis, Politecnico Di Milano, Italy, 2008.

[94] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *Computer*, 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1193228.

[95] Kun Lu, Daniel Müller-Gritschneder, Ulf Schlichtmann. Accurately Timed Transaction Level Models for Virtual Prototyping at High Abstraction Level. In *DATE*, 2012.

[96] R. Leupers, F. Schirrmeister, G. Martin, T. Kogel, R. Plyaskin, A. Herkersdorf, M. Vaupel. Virtual platforms: Breaking new grounds. In *DATE*, 2012.

[97] A. Gerstlauer, S. Chakravarty, M. Kathuria, P. Razaghi. Abstract system-level models for early performance and power exploration. In *The 17th Asia and South Pacific Design Automation Conference*, 2012.

[98] Zai Jian Jia, Tomás Bautista, Antonio Núñez, Andy D. Pimentel, and Mark Thompson. A system-level infrastructure for multidimensional mp-soc design space co-exploration. *ACM Trans. Embed. Comput. Syst.*, 2013.

[99] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 2011.

[100] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, M. Wouters, C. Kavka, L. Onesti, A. Turco, U. Bondik, G. Mariani, H. Posadas, E. Villar, C. Wu, F. Dongrui, Z. Hao, and T. Shibin. Multicube: Multi-objective design space exploration of multi-core architectures. In *2010 IEEE Computer Society Annual Symposium on VLSI*, 2010.

[101] V. Zaccaria, G. Palermo, F. Castro, C. Silvano, and G. Mariani. Multicube explorer: An open source framework for design space exploration of chip multiprocessors. In *23th International Conference on Architecture of Computing Systems 2010*, 2010.

[102] modeFRONTIER. http://www.esteco.com/modefrontier, 2018.

[103] COMPLEX: an EU funded project. http://owsgip.itc.utwente.nl/projects/complex, 2018.

[104] K. Grüttner, P. A. Hartmann, K. Hylla, S. Rosinger, W. Nebel, F. Herrera, E. Villar, C. Brandolese, W. Fornaciari, G. Palermo, C. Ykman-Couvreur, D. Quaglia, F. Ferrero, and R. Valencia. Complex: Codesign and power management in platform-based design space exploration. In *2012 15th Euromicro Conference on Digital System Design*, 2012.

[105] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 2002.

[106] M. Mefenza, F. Yonga, L. B. Saldanha, C. Bobda, and S. Velipassalar. A framework for rapid prototyping of embedded vision applications. pages 1–8, 2014.

[107] W. Zuo, W. Kemmerer, J. B. Lim, L. N. Pouchet, A. Ayupov, T. Kim, K. Han, and D. Chen. A polyhedral-based systemc modeling and generation framework for effective low-power design space exploration. pages 357–364, 2015.

[108] T. R. Mück and A. A. Froehlich. Seamless integration of hw/sw components in a hls-based soc design environment. pages 109–115, 2013.

[109] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels. A survey and evaluation of fpga high-level synthesis tools. pages 1591–1604, 2016.

[110] mentor. Catapult HLS. https://www.mentor.com/hls-lp/catapult-high-level-synthesis/, 2016.

[111] A. Canis, J. Choi, B. Fort, R. Lian, Q. Huang, N. Calagar, M. Gort, J. J. Qin, M. Aldham, T. Czajkowski, S. Brown, and J. Anderson. From software to accelerators with legup high-level synthesis. pages 1–9, 2013.

[112] ITRS. International technology roadmap for semiconductors. 2009.

[113] Jinja2. http://jinja.pocoo.org/docs/2.10/, 2018.

[114] B. Farkas, S. A. A. Shah, J. Wagner, R. Meyer, R. Buchty, and M. Berekovic. An open and flexible systemc to vhdl workflow for rapid prototyping. In *"Design and Verification Conference (DVCon) Europe*, 2016.

[115] T. Givargis, F. Vahid, and J. Henkel. System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, 2001.

[116] Nuwan I. Senaratna. Genetic algorithms: The crossover-mutation debate. 2005.

[117] EL-Sayed M. Zaki A. A. Mousa Ahmed A. EL-Sawy, Mohamed A. Hussein. An introduction to genetic algorithms: A survey a practical issues. 2014.

[118] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[119] Adam Marczyk. Genetic Algorithms and Evolutionary Computation . http://www.talkorigins.org/faqs/genalg/genalg.html, 2004.

[120] Kumara Sastry and David E. Goldberg. Analysis of mixing in genetic algorithms: A survey. Technical report, 2002.

[121] Onur Boyabatli and Ihsan Sabuncuoglu. Parameter selection in genetic algorithms. *Journal of Systemics, Cybernetics and Informatics*, page 78, 2004.

[122] J. Teich. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, pages 1411–1430, 2012.

[123] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Y. Wei, and D. Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016.

[124] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel. Capi: A coherent accelerator processor interface. *IBM Journal of Research and Development*, pages 7:1–7:7, 2015.

[125] George Kyriazis. Heterogeneous system architecture: A technical review. *AMD Fusion Developer Summit*, 2012.

[126] Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, 1984.

[127] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. 2001.

[128] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 2008.

[129] Michael Grabner, Helmut Grabner, and Horst Bischof. Fast approximated sift. In *Computer Vision – ACCV 2006*. Springer Berlin Heidelberg, 2006.

[130] Marcos Slomp and Manuel Menezes de Oliveira. Real-time photographic local tone reproduction using summed-area tables. 2008.

[131] E. Magli. Multiband Lossless Compression of Hyperspectral Images. *IEEE Transactions on Geoscience and Remote Sensing*, 47, 2009.

[132] The Consultative Commitee for Space Data Systems. Lossless multispectral and hyperspectral image compression, draft recommendation for space data system standards. 2011.

[133] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, IISWC '09. IEEE Computer Society, 2009.

[134] Real-Time Executive for Multi-Processor Systems (RTEMS). http://www.rtems.org, 2017.

[135] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *WWC-4. 2001 IEEE International Workshop on*, 2001.